Universidad
Carlos III de Madrid

Grado en ingeniería de sistemas audiovisuales
Trabajo de Fin de Grado

Desarrollo de un sistema de comunicaciones basado en WebRTC

Autor: Federico Martín Sánchez
Tutor: Iván Vidal Fernández
Leganés, 2016

"El verdadero progreso es el que pone la tecnología al alcance de todos"

Henry Ford (1863-1947)

AGRADECIMIENTOS

Quisiera dar las gracias a mis compañeros de carrera, por hacer este viaje más ameno y no dejarme olvidar que lo más importante es disfrutar aprendiendo.

Gracias a mi tutor Iván por su comprensión, disponibilidad y paciencia.

Y gracias a mis padres por su apoyo incondicional en todo momento y sus consejos que me hacen ser cada día un poco más sabio.

RESUMEN

Internet está cambiando el mundo tal y como lo conocemos en muy poco tiempo. En particular, el sector de las telecomunicaciones está constantemente reinventándose. Son proyectos como WebRTC los que, aunque empiezan teniendo poca repercusión, traen consigo una enorme ambición y consiguen ser el impulso de estos cambios.

En concreto la motivación de la tecnología WebRTC es crear una web más dinámica, introduciendo comunicaciones en tiempo real en los navegadores tradicionales sin necesidad de instalaciones ni complementos de terceros. Esto puede suponer una gran innovación para las plataformas multimedia en Internet, como servicios de video chat o plataformas de streaming de video, que cada vez tienen más presencia en la vida cotidiana de los usuarios

El principal objetivo de este proyecto es el estudio teórico y práctico de esta tecnología, desarrollando y desplegando una plataforma de videoconferencias para múltiples participantes que funcione en el navegador, y que permita conocer desde la perspectiva de un desarrollador, el estado actual y las posibilidades de WebRTC, permitiéndonos llegar a una concusión sólida sobre su madurez y el posible impacto que pueda tener en el mundo tecnológico actual.

ABSTRACT

Internet is changing the world as we know it in a very short period of time. In particular, the telecom industry is constantly reinventing itself. Are the projects like WebRTC which, although they start having little repercussion, bring a huge ambition and get to be the engine of these changes.

Primarily, the motivation of the WebRTC project is bringing a more dynamic web, introducing real time communications in traditional browsers without software installations or third party plugins. This can be a huge innovation for multimedia platforms on the Internet like video chat services or online video streaming, which are significantly increasing their presence in people daily lives.

The main objective of this project is the theoretical and practical study of this technology, developing and deploying a videoconferencing application for multiple participants that works in the browser, which allows to know from the perspective of a web developer, the current situation and potential of WebRTC allowing us to reach a solid conclusion on its maturity and the possible impact it may have in today's tech world.

INFORMACIÓN COMPLEMENTARIA

El presente Trabajo de Fin de grado cuenta con cuatro anexos:

- El **Anexo A** contiene la planificación y el presupuesto total del proyecto.

- Los **Anexos B, C y D** contienen la introducción, conclusión y resumen del proyecto redactados en inglés acorde a la Normativa de los alumnos de Grado del Plan 2011.

ÍNDICE

INDICE DE FIGURAS

GLOSARIO DE ACRÓNIMOS

- **API** Application Programming Interface
- **APP** Application
- **CSS** Cascading Style Sheets
- **HTML** Hypertext Markup Language
- **HTTP** Hypertext Transfer Protocol
- **ICE** Interactive Connectivity Establishment
- **iLBC** Internet Low Bit Rate Codec
- **IP** Internet Protocol
- **iSAC** Internet Speech Audio Codec
- **JS** JavaScript
- **NAT** Network Address Translation
- **P2P** Peer-to-Peer
- **PC** Peer Connection
- **RTC** Real Time Communications
- **RTCP** Real-Time Control Protocol
- **RTCP** RTP Control Protocol
- **RTCPC** RTCPeerConnection
- **RTP** Real-time Transport Protocol
- **SCTP** Stream Control Transmission Protocol
- **SDP** Session Description Protocol
- **SIP** Session Initiation Protocol
- **SRTP** Secure Real-Time Transport Protocol
- **STUN** Session Traversal Utilities for Nat
- **TCP** Transmission Control Protocol
- **TURN** Traversal Using Relays around Nat
- **UDP** User Datagram Protocol
- **URI** Uniform Resource Identifier
- **URL** Uniform Resource Locator
- **WWW** World Wide Web

1. Introducción

1.1. MOTIVACIÓN

Es difícil percibir la velocidad con la que cambia la tecnología. Hace 20 años los teléfonos móviles eran una revolución y disponer de una conexión a internet en los hogares no era algo tan común. Hoy en día, la mayoría de personas cuenta con varios dispositivos conectados internet de forma continua, lo cual ha supuesto una constante evolución en las co*mun*icaciones a distancia, que poco a poco van pasando de las llamadas telefónicas tradicionales a ser en su mayor parte a través de internet.

En mayo de 2011 Google presentaba un proyecto de código libre conocido como WebRTC [1], que pretendía introducir comunicaciones en tiempo real en el navegador cambiando la forma en la que percibimos la 'web'. El hecho de poder conectar navegadores e intercambiar flujos de información de audio y video supone una gran innovación respecto a la industria de telefonía y conferencia audiovisual. Pero esto es solo el principio, la capacidad para los desarrolladores JavaScript de poder trabajar con la información de la cámara y el micrófono del equipamiento de usuario, así como con otros medios que pueden estar disponibles en el mismo (ej. pantalla o ficheros en el sistema local), abre un nuevo abanico de posibilidades y crea una web más dinámica que permite a las aplicaciones interactuar con los usuarios en tiempo real por medio de voz, gestos, y muchas nuevas opciones.

Hasta ahora podíamos obtener algo parecido a este tipo de funcionalidades en el navegador con complementos como flash, pero esto traía una serie de problemas como cierres forzosos o actualizaciones constantes, que desembocaban en una percepción de baja calidad por parte del usuario.

Por otro lado, también resulta complicado hablar de videoconferencias y no pensar en Skype, *¿Qué ventajas puede presentar WebRTC frente a Skype?*

En primer lugar, Skype es una aplicación, es un programa que requiere una instalación en un equipo y nos ofrece unas funcionalidades definidas. WebRTC sin embargo es un conjunto de herramientas a disposición de todos, que permite a los programadores web aprovechar sus conocimientos de los principales lenguajes de

programación web (HTML, CSS y JavaScript) para añadir funcionalidades de comunicaciones multimedia a sus creaciones.
De primeras, esto nos presenta dos grandes ventajas:

- Las aplicaciones basadas en WebRTC no requieren de ninguna descarga o instalación por parte del usuario, simplemente abrir la página desde el navegador.
- El hecho de que sea una tecnología de código libre, da rienda suelta a la creatividad de los desarrolladores para crear nuevas ideas. Las aplicaciones de esta tecnología en un futuro cercano pueden significar un paso adelante en campos desde los videojuegos online hasta sistemas de educación o medicina a distancia pasando por revolucionar las plataformas de streaming en el navegador.

En la línea de lo argumentado anteriormente, la motivación para este trabajo ha sido explorar el potencial de la tecnología WebRTC para soportar el desarrollo de aplicaciones de comunicación multimedia. En el siguiente apartado de describe en detalle este objetivo principal, así como todos los sub-objetivos que se derivan del mismo.

1.2. OBJETIVOS

El objetivo principal de este TFG es analizar, desde una perspectiva práctica, el soporte proporcionado por la tecnología WebRTC para diseñar, desarrollar y desplegar aplicaciones de comunicación multimedia. Para ello, se propone el desarrollo de una aplicación de conferencia, que posibilite el intercambio en tiempo real de audio, vídeo y mensajes de texto entre múltiples usuarios que participen en la misma.

En línea con este objetivo principal, se definen los siguientes sub-objetivos:

- Analizar el estado del arte sobre esta tecnología, revisando la información técnica y actualizada sobre el funcionamiento de WebRTC, su arquitectura, protocolos, estándares y herramientas, así como las implementaciones más populares.
- Valorar las posibilidades reales para desarrollar un producto basado en esta tecnología, considerando la dificultad,

obstáculos y facilidades que se nos presentan en el panorama actual.
- Probar y evaluar el rendimiento de las aplicaciones WebRTC.
- Explorar los casos de uso potenciales de esta tecnología.
- Llegar a una conclusión sólida y hacer una valoración de la utilidad y madurez de la tecnología WebRTC.

1.3. ESTRUCTURA DE LA MEMORIA

A continuación se describe brevemente el contenido de cada capítulo que compone la memoria de este TFG.

Capítulo 1. Introducción: En el presente capítulo se introducen las motivaciones que han conducido a la elaboración de este trabajo, así como los objetivos del mismo y la estructura del documento, con una breve descripción del contenido de cada capítulo.

Capítulo 2. Estado del Arte de WebRTC: En este capítulo se explica el funcionamiento y características de la tecnología WebRTC desde una perspectiva técnica, descubriendo sus limitaciones y bondades actuales y las arquitecturas más populares.

Capítulo 3. Requisitos de la Plataforma: En este capítulo se enumeran los requisitos funcionales que pretende tener la aplicación web desarrollada para este proyecto y los requisitos de usuario necesarios para utilizarla.

Capítulo 4. Implementación: Dividiendo la plataforma en dos bloques: Aplicación cliente y servidor de conferencia, este capítulo detalla el diseño y funcionamiento de la aplicación, apoyándose en capturas de pantalla y diagramas de flujo. También se describen en este capítulo los recursos y herramientas utilizados para el desarrollo.

Capítulo 5. Validación de la aplicación: Este capítulo está dedicado a hacer pruebas con la aplicación, mostrando test de uso y comprobando el rendimiento de la plataforma.

Capítulo 6. Conclusiones: Este último apartado está destinado a hacer una valoración general de la tecnología, comentar su nivel de madurez y su utilidad de cara a un futuro cercano.

2. ESTADO DEL ARTE DE WEBRTC

2.1. ARQUITECTURA DE WEBRTC

En este apartado se explican los principales componentes de la arquitectura general de WebRTC:

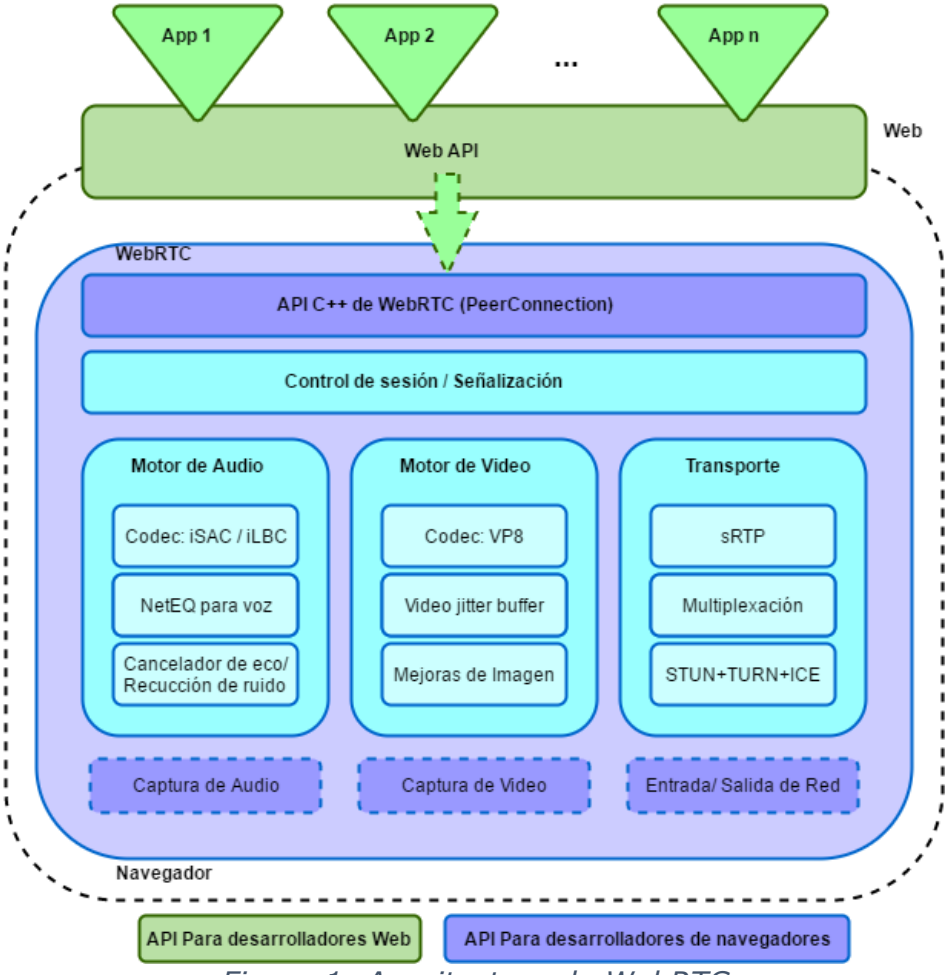

Figura 1: Arquitectura de WebRTC

En la Figura 1 se pueden diferenciar dos capas principales: La capa orientada a los desarrolladores Web, y la capa orientada a los desarrolladores de navegadores.

2.1.1. API para desarrolladores de Web

Cualquier desarrollador web puede hacer uso de la Web API para desarrollar aplicaciones de comunicación de audio y video en el navegador sin necesidad de complementos o instalaciones ajenas.

Esta API está definida por el W3C y la documentación de su última versión se puede encontrar en el siguiente enlace: *https://www.w3.org/TR/webrtc/*
Se analizan las principales herramientas de esta API en el apartado 1.3.

2.1.2. API para desarrolladores de Navegadores

Los desarrolladores de navegadores estarán interesados en la API nativa de WebRTC C++, que utilizarán para dotar a sus navegadores de las capacidades necesarias para correr aplicaciones basadas en WebRTC.

En la arquitectura de WebRTC también se definen los Motores de audio y video, y el Transporte de contenido multimedia. En los siguientes sub-apartados se describen los codecs que se utilizan en los motores de Audio y Video.

2.1.2.1. Motor de Audio

Actualmente WebRTC soporta los siguientes codecs de audio:

- **OPUS** [2]**:** Lanzado en 2012, este códec se utiliza en Firefox y Chrome por defecto para codificar streams de audio. Soporta tasas de bit de 6 a 510 kbit/s y tasas de muestreo de 89 a 48 kHz.

- **iSAC (Internet Speech Audio Codec)** [3]**:** En el pasado iSAC era software propietario desarrollado para aplicaciones VoIP y streaming de audio. Fue adquirido por Google en 2011 y desde entonces empezó a formar parte del proyecto WebRTC.
  Soporta tasas de bit de 10 a 52 kbit/s y tasa de muestreo de 32kHz.
  Es apropiado para el intercambio de para voz, pero no recomendable para flujos de audio de alta calidad.

- **iLBC (Internet Low Bit Rate Codec)** [4]**:** También adquirido por Google en 2011, este códec es útil en situaciones donde se dispone de canales de mala calidad y ancho de banda bajo. Soporta tasas de bit de 15.2 a 1.33 kbit/s y tasa de muestreo de 8 kHz

2.1.2.2. Motor de Video

La codificación de video en WebRTC es bastante limitada, soportando únicamente VP8 [5] y H.264 [6], éste último solo disponible en Firefox.

Estas limitaciones en la codificación de video en WebRTC recortan el potencial de esta tecnología para aplicaciones de streaming de video, que están teniendo un impacto creciente en los últimos años con plataformas como Netflix.

El desarrollo de WebRTC en este aspecto se mantiene activo, por ejemplo, a partir de la versión 48 de Chrome, VP9 [7] está disponible como códec opcional para las llamadas WebRTC[1].

2.1.2.3. Transporte

La principal función del transporte en WebRTC es el intercambio de información multimedia en tiempo real entre extremos, para ello se utiliza el protocolo sRTP (Secure Real-Time Control Protocol), del que se habla en más profundidad en el apartado 2.3.2.

Para que la sesión RTP pueda llevarse a cabo, también será necesario utilizar otros protocolos de transporte para intercambiar información de red y datos de configuración de la sesión multimedia entre los participantes. Se habla de estas funciones con detalle en los siguientes apartados.

2.1.2.4. Navegadores y plataformas soportadas

Una de las limitaciones más importantes de WebRTC es el hecho de que las plataformas soportadas son bastante escasas. Navegadores muy populares como Safari, el navegador por defecto en todos los Mac, o la mayoría de navegadores de dispositivos móviles no soportan aún

[1] Fuente: https://developers.google.com/web/updates/2016/01/vp9-webrtc

WebRTC. En la Figura 2 se recoge información actualizada sobre el soporte de WebRTC para los navegadores más populares.

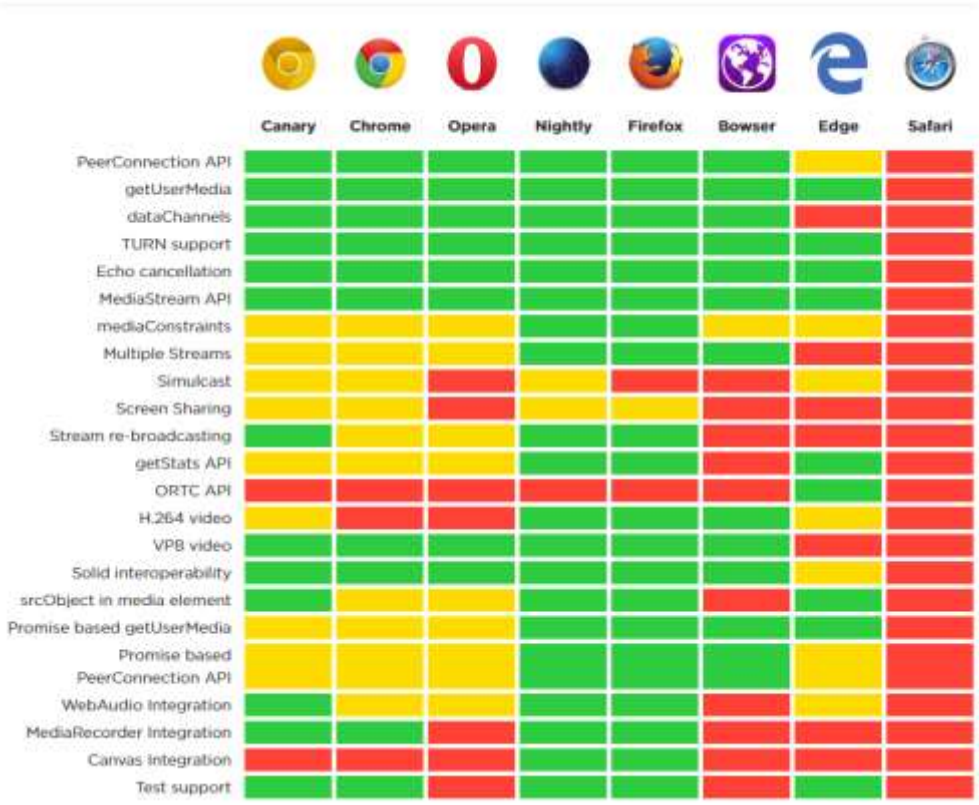

Figura 2: Tabla de soporte de WebRTC en los navegadores más populares2

[2] Tabla obtenida de la página http://iswebrtcreadyyet.com/

2.2. APIS PRINCIPALES

Existen tres tareas principales para desarrollar una aplicación basada en WebRTC:

- Obtener audio y video de la cámara y el micrófono del usuario y asociarlo a un elemento HTML.
- Intercambiar audio y video entre los participantes de la sesión.
- Intercambiar otra información (datos ajenos a la sesión audiovisual)

Cada una de estas tareas se realiza mediante una API Javascript específica. En los siguientes apartados se explican con detalle las tres APIs principales de WebRTC.

2.2.1. MediaStream

La API MediaStream [8] facilita el acceso a los flujos de información procedentes de la cámara y el micrófono local.
Un objeto MediaStream es típicamente una URL de referencia a una información multimedia que se almacena en un archivo o un objeto BLOB (Binary Large Object) creado con el método `window.url.createObjectURL()`. Los objetos MediaStream contienen a su vez varios objetos MediaStreamTrack, los cuales representan video y audio de diferentes dispositivos. A su vez los objetos MediaStreamTrack pueden incluir varios canales (canales L y R de audio).

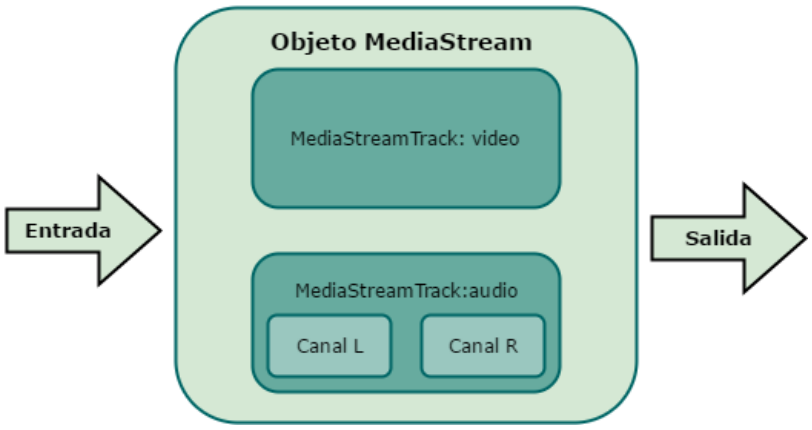

Figura 3: Representación de un Objeto MediaStream

Para obtener el objeto MediaStream y asociarlo a un elemento HTML se utiliza la llamada al método **`getUserMedia()`**, a continuación se ilustra el procedimiento a través de un breve fragmento de código JavaScript:

```
var constraints = {video: true, audio: true};

function exito(stream) {
var video = document.querySelector("video");
video.src = window.URL.createObjectURL(stream);
}

function error(error) {
console.log("Error obteniendo el video local: ", error);
}

navigator.getUserMedia(constraints, exito, error);
```

Figura 4: funciones principales de getUserMedia()

La variable constraints define el contenido del objeto MediaStream, si es audio o video, la resolución, la tasa de imágenes por segundo y, en caso de contar con varias cámaras de video o varios micrófonos, el dispositivo a utilizar.

La función exito() define los pasos a seguir en caso de que la llamada a getUserMedia() se realice correctamente. Se crea una variable que se asocia a un elemento de video HTML y posteriormente se le adjunta el stream obtenido.

La función error() se llama en caso de que la llamada a getUserMedia falle y muestra por consola el error.

2.2.2. RTCPeerConnection

La API RTCPeerConnection [1] es el componente principal de WebRTC, ya que representa la conexión del entre el ordenador local y los Peers que participan en la sesión. Contiene métodos para conectarse a ordenadores remotos, monitorizar la conexión, y cerrar la conexión en el momento deseado.

Como se puede observar en la figura del apartado 1.2 (Arquitectura de WebRTC), RTCPeerConnection se encarga de bastantes tareas que simplifican en gran medida el trabajo de los desarrolladores web, como procesamiento de señales, manejo de codecs, comunicación entre Peers, seguridad, gestión del ancho de banda…

A continuación, se muestra un breve fragmento de código JavaScript que ilustra las funciones principales de un objeto RTCPeerConnection:

```
pc = new RTCPeerConnection(null);
pc.onaddstream = gotRemoteStream;
//Cuando se añade un stream remoto a la conexion, se llama al método gotRemoteStream()
pc.addStream(localStream);
//Se añade el Stream local obtenido con getUserMedia() a la conexión
pc.createOffer(gotOffer);//método para crear la oferta SDP

function gotOffer(desc) {
  pc.setLocalDescription(desc);//construye el SDP
  sendOffer(desc);// lo envía por el canal de señalización
}

function gotAnswer(desc) {
  pc.setRemoteDescription(desc);
//Asocia el SDP entrante como respuesta SDP en la conexión
}

function gotRemoteStream(e) {
  attachMediaStream(remoteVideo, e.stream);
//asocia el stream remoto a un elemento HTML para poder visualizarlo
}
```

Figura 5: Funciones principals de RTCPeerconnection

Este fragmento de código representa el intercambio SDP y la función para asociar el stream remoto a la variable 'remoteVideo' en el HTML.

2.2.3. RTCDataChannel

La API RTCDataChannel [9] representa un canal bidireccional de intercambio de datos entre Peers. Esta herramienta aumenta en gran medida la utilidad de WebRTC, pudiendo extenderse a aplicaciones como videojuegos online o transferencia de archivos.

Actualmente hay una gran variedad de opciones disponibles para crear un canal de comunicaciones (WebSocket, AJAX…), pero mientras estas tecnologías están diseñadas para utilizar un servidor como intermediario, RTCDataChannel funciona con conectividad Peer-to-Peer, lo cual aporta una serie de ventajas como latencias muy bajas o

la utilización del protocolo de transporte SCTP, del que se habla con más detalle en el apartado 2.3.6.

2.3. PROTOCOLOS

2.3.1. HTTP

HTTP (Hyper-text Transport Protocol) [10] es el protocolo de la World Wide Web, utilizado en la comunicación entre un navegador y un servidor web.
WebRTC utiliza este protocolo como cualquier aplicación web, por tanto, no se requiere un conocimiento demasiado específico de este protocolo.
HTTP sigue un modelo petición respuesta. El cliente realiza una petición enviando un mensaje determinado y el servidor envía un mensaje de respuesta.

En la Figura 6 podemos ver un ejemplo de diálogo HTTP en la que el cliente envía una petición GET a la URI *"www.example.com/hello.txt"*

```
Client request:

  GET /hello.txt HTTP/1.1
  User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
  Host: www.example.com
  Accept-Language: en, mi


Server response:

  HTTP/1.1 200 OK
  Date: Mon, 27 Jul 2009 12:28:53 GMT
  Server: Apache
  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
  ETag: "34aa387-d-1568eb00"
  Accept-Ranges: bytes
  Content-Length: 51
  Vary: Accept-Encoding
  Content-Type: text/plain

  Hello World! My payload includes a trailing CRLF.
```

Figura 6: Ejemplo de diálogo HTTP

2.3.2. RTP y sRTP

RTP (Real-time Transport Protocol) [11] es el protocolo más importante que se utiliza en WebRTC, pues es el encargado de transportar los paquetes de audio y video entre los participantes de la sesión.

RTP ofrece un servicio de entrega extremo a extremo en tiempo real. Su función básica es la de multiplexar varios flujos de información multimedia en un solo flujo de paquetes UDP, que pueden enviarse a un solo destino (unicast) o a varios (multicast). Los paquetes están numerados de forma ascendente a través de un número de secuencia, de forma que la aplicación pueda conocer la situación de los paquetes que fallen. Se ejecuta, en general, sobre UDP, ya que es preferible para aplicaciones multimedia sacrificar la fiabilidad de TCP a cambio de mayor velocidad y menores retardos.

Otra característica importante de RTP son las marcas temporales (time-stamping) que introduce el origen, son relativas al inicio del flujo y sirven para que el destino pueda almacenar un buffer e ir reproduciendo cada muestra en el momento exacto después del inicio del flujo, reduciendo así los cortes y sincronizando los múltiples flujos de información multimedia. En la Figura 8 podemos observar los campos de la cabecera RTP con una pequeña descripción de cada uno.

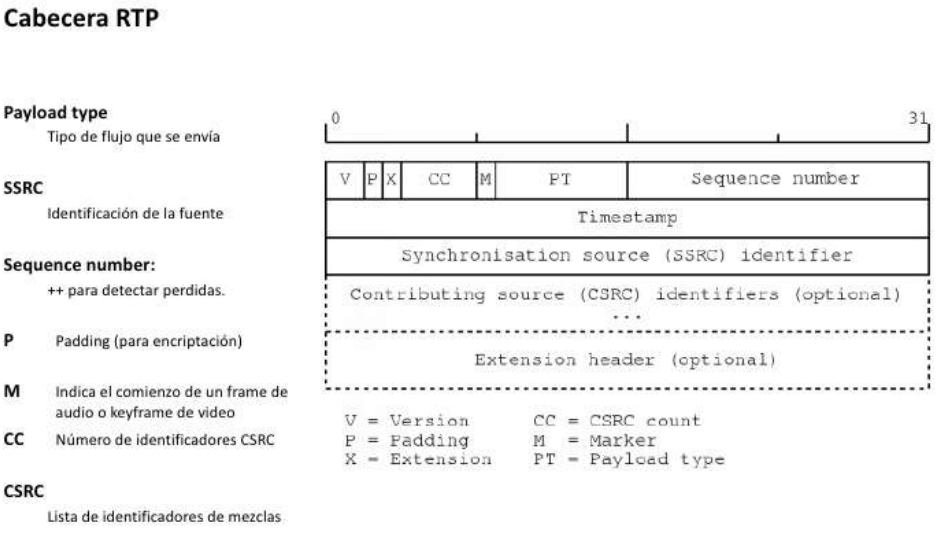

Figura 7: Cabecera RTP3

[3] Fuente: http://es.slideshare.net/jcague/rtp-y-sdp

El protocolo RTCP (Real-Time Control Protocol) es complementario a RTP y le aporta un mecanismo de control. Se basa en el envío de paquetes de control a los participantes de la sesión que contienen información de la calidad de los datos emitidos por la fuente. También se ejecuta sobre UDP, por el puerto siguiente al que se utiliza para RTP.

WebRTC utiliza la versión segura de RTP, sRTP [12] (Secure Real-Time Transport Protocol), que incorpora características de seguridad como el cifrado, autenticación de mensajes e integridad, y protección contra reenvíos a los datos RTP en aplicaciones unicast y multicast.

2.3.3. SDP

El protocolo SDP [13](Session Description Protocol) se utiliza para describir las características multimedia de la conexión entre los Peers. Hay una larga lista de información que debe ser intercambiada entre los participantes para poder establecer la conexión sRTP y el intercambio de paquetes multimedia.

El intercambio de paquetes SDP sigue un modelo de oferta/respuesta, y la información que contienen está dividida en líneas, cada una de ellas describiendo un atributo. En la Figura 8 se observa el formato SDP y las líneas más importantes.

```
Descripción de la sesión
   v=  (Versión del protocolo)
   o=  (Origen e identificador de sesión)
   s=  (Nombre de sesión)
   i=* (Información de la sesión)
   u=* (URI de descripción)
   e=* (Correo electrónico)
   p=* (Número telefónico)
   c=* (Información de conexión)
   b=* (Cero o más líneas con información de ancho de banda)
   Una o más líneas de descripción de tiempo (Ver abajo "t=" y "r=")
   z=* (Ajustes de zona horaria)
   k=* (Clave de cifrado)
   a=* (Cero o más líneas de atributos de sesión)
   Cero o más descripciones de medios

Descripción de tiempo
   t=  (Tiempo durante el cual la sesión estará activa)
   r=* (Cero o más veces de repetición)

Descripción de medios, si está presente
   m=  (Nombre de medio y dirección de transporte)
   i=* (Título)
   c=* (Información de conexión)
   b=* (Cero o más líneas con información de ancho de banda)
   k=* (Clave de cifrado)
   a=* (Cero o más líneas de atributos de sesión)
```

Figura 8: Formato de SDP

2.3.4. WebSocket

El protocolo WebSocket [14] tiene como objetivo facilitar a los desarrolladores web establecer una conexión TCP bidireccional entre el Navegador y un servidor web.

Los beneficios de este protocolo son muchos, ya que se puede conseguir un intercambio de mensajes cliente/servidor sin retardos y con tráfico de red muy ligero. Esto ha conseguido que la mayoría de desarrolladores de aplicaciones WebRTC utilicen WebSockets como solución principal para las funciones de señalización.

La API de WebSockets está disponible en la mayoría de lenguajes de programación. En el caso de WebRTC la implementación más popular es junto a ***Node.js*** [15] ya que está basado en JavaScript. Librerías como ***Socket.io*** [16], hacen muy sencilla la implementación de este protocolo.

2.3.5. SCTP

SCTP (Stream Control Transport Protocol) [17] es un protocolo de la capa de transporte que se utiliza en WebRTC a través de la API RTCDataChannels.
Este protocolo proporciona ventajas sobre los protocolos de transporte tradicionales (TCP y UDP) ya que provee control de congestión, secuenciación y confiabilidad como TDP, y al mismo tiempo nos permite de forma opcional el envío de mensajes fuera de orden. También cuenta con soporte para múltiples flujos de información en una sesión.

En la figura 9 se puede observar una tabla comparativa de funcionalidades/características entre sCTP, TCP y UDP. Pudiéndose comprobar que sCTP combina las características destacadas de UDP y TDP añadiendo algunas como multiplexación, soporte para multi-homing y multi-streaming, mensajes de seguridad y mensajes heartbeat para controlar el estado de actividad de una aplicación.

CARACTERISTICAS DE PROTOCOLO	SCTP	TCP	UDP
Estado almacenados en los terminales	Si	Si	No
Transferencia confiable de los datos	Si	Si	No
Control de congestion	Si	Si	No
Delimitación de límites en los mensajes	Si	No	Si
Fragmentación e integración de la información	Si	Si	No
Multiplexación de información del paquete	Si	Si	No
Soporte de *multi-homing*	Si	No	No
Soporte de *multi-streaming*	Si	No	No
Envío de datos desordenado	Si	No	Si
Cookie de seguridad para evitar ataques de inundación de SYN	Si	No	No
Mensaje *heartbeat*	Si	No	No

Figura 9: Tabla comparativa entre SCTP, TCP y UDP[4]

[4] Fuente: http://profesores.elo.utfsm.cl/~agv/elo323/2s10/projects/CaponaAhumada/comparacion.html

2.4. ICE

Es bastante común encontrarnos con que los navegadores están detrás de un NAT (Network Address Translation, Traductor de direcciones de red), que normalmente se usa para permitir a varios dispositivos compartir una misma dirección IP. Esto no supone una dificultad para muchos de los protocolos populares de internet. Sin embargo, para los protocolos y servicios Peer-to-peer como es el caso de WebRTC, esto puede suponer una dificultad importante.

Las aplicaciones WebRTC utilizan el protocolo ICE para solucionar estas dificultades.
ICE (Interactive Connectivity Estabishment) [18] es un protocolo que ayuda a las aplicaciones multimedia basadas en el modelo de oferta/respuesta a traspasar los NAT. Para ello utiliza dos protocolos principales; **STUN y TURN**.

2.4.1. STUN

El protocolo de red STUN (Session Traversal Utilities for NAT) [19] envía paquetes de test previos al establecimiento de la sesión multimedia para permitir al navegador saber si está detrás de un NAT y obtener su dirección IP desde una perspectiva pública.

En la figura 10 se representa el funcionamiento del protocolo STUN. Dos dispositivos están detrás de un NAT y necesitan saber su dirección IP pública para poder establecer una conexión RTP, por tanto, enviarán una petición a un servidor STUN, el cual responderá con la IP y puerto del dispositivo desde una perspectiva pública.

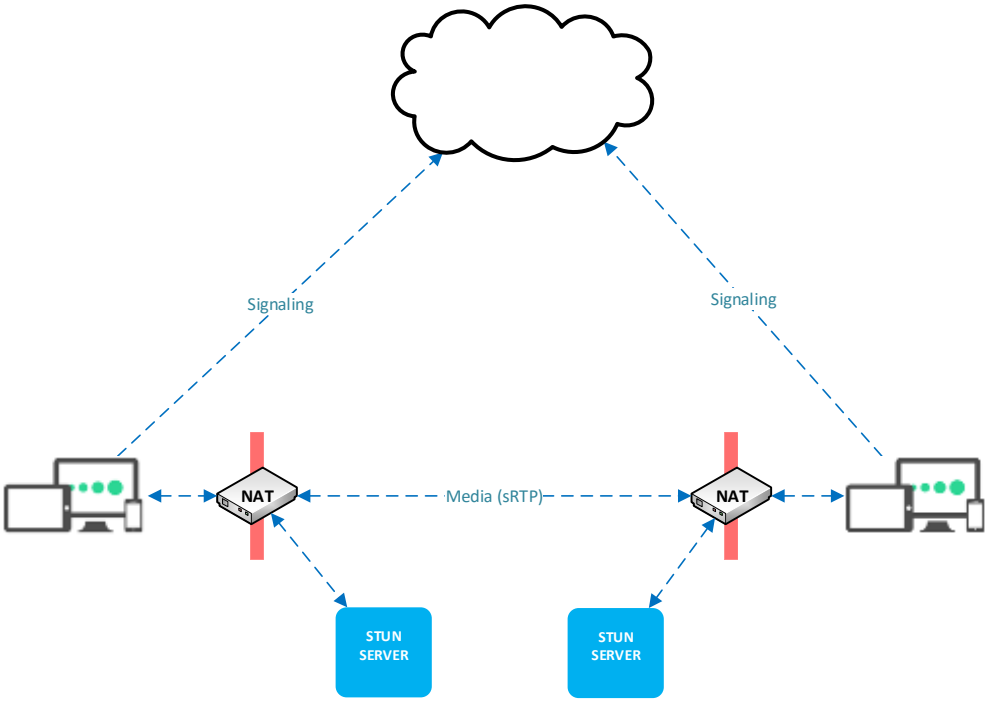

Figura 10: Escenario utilizando un servidor STUN para obtener una IP Pública

2.4.2. TURN

El protocolo TURN (Traversal Using Relays around NAT) [20] es una extensión de STUN. Se utiliza para retransmitir información entre los Peers si RTCPeerConnection falla en el establecimiento de la comunicación entre estos.

En algunos casos, podemos encontrar firewalls muy restrictivos que no permitan el intercambio de candidatos ICE, o incluso bloqueen el trafico multimedia (UDP) entre los Peers. Una solución en este caso es incluir un servidor TURN permitido por el firewall que retransmita la información multimedia.

Todo servidor TURN puede ser utilizado como un servidor STUN con la funcionalidad añadida de retransmisión de tráfico multimedia entre los Peers.

En la figura 11 se representa el funcionamiento de un servidor TURN cuando el servidor STUN falla. Los extremos no pueden conocer su dirección IP Pública, por lo que la sesión multimedia (RTP) se establece entre dos servidores TURN que actúan como intermediarios retransmitiendo el tráfico de paquetes de audio y video.

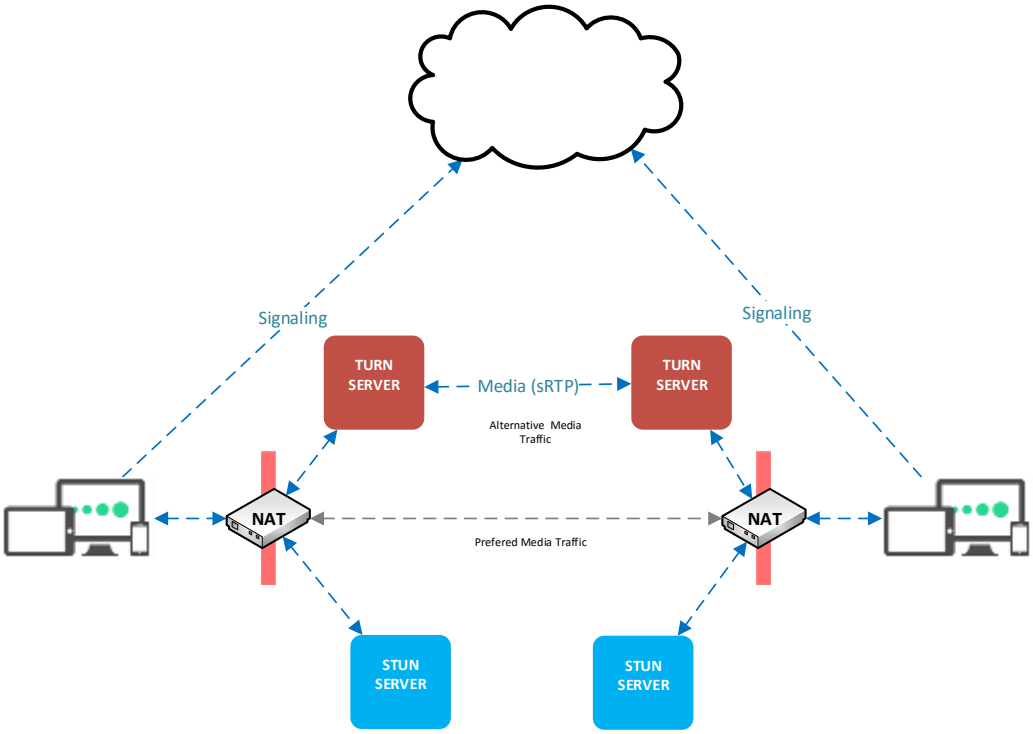

Figura 11: Si STUN falla, se puede utilizar TURN para transportar los paquetes multimedia

2.5. SEÑALIZACIÓN

La señalización en WebRTC juega un papel muy importante, el de establecer un canal que permita intercambiar mensajes entre los participantes. Sin embargo, esta función no está estandarizada, permitiendo al desarrollador escoger los protocolos que considere convenientes.
El rol principal de la señalización se resume en estos cuatro puntos:

- Negociación de la sesión multimedia
- Identificación y autentificación de los participantes en la sesión (opcional)
- Control de la sesión, indicando el progreso, y terminándola
- Intercambio de candidatos (direcciones y puertos para enviar/recibir información)

En los apartados siguientes se describirán estos puntos con más detalle y se dedicará un apartado final en el que se hablará brevemente sobre las opciones disponibles y su viabilidad en la actualidad.

Las funciones de señalización en WebRTC siguen un modelo definido por el protocolo JSEP [21] (JavaScript Session Establishment

Protocol). El plano de señalización se deja a la aplicación, ya que dependiendo de la aplicación será conveniente utilizar determinados protocolos para llevar a cabo esta función.
De esta forma se exime al navegador de tener que guardar el estado de señalización. Esto sería problemático, ya que cada vez que se actualizase la página en el navegador se perdería el estado de señalización.

Para que se pueda llevar a cabo la sesión multimedia, JSEP requiere que se complete un intercambio de oferta/respuesta SDP entre los extremos. La arquitectura de JSERP se puede ver en la Figura 12 a continuación.

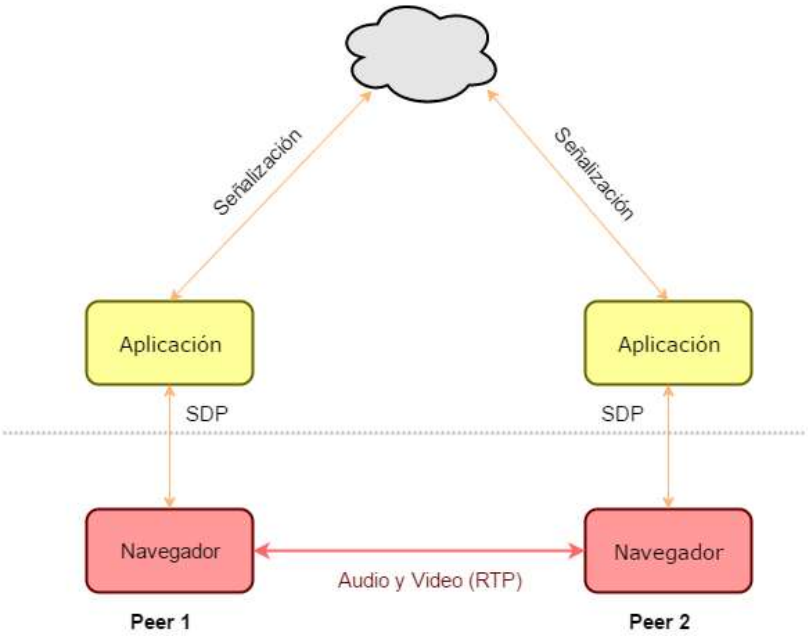

Figura 12: Señalización en WebRTC. Arquitectura de JSEP

En la Figura 12, el Peer1 está tratando de llamar al Peer2, a continuación, se enumeran los pasos detallados que debería llevar a cabo:

1. Peer1 crea un objeto `RTCPeerConnection`.
2. Peer1 crea la oferta SDP mediante el método `createOffer()` de `RTCPeerConnection`.
3. Peer1 llama al método `setLocalDescription()`.
4. Peer1 utiliza un mecanismo de señalización para hacer llegar la oferta a Peer2

5. Peer2 llama al método `setRemoteDescription()` con la oferta de Peer1, para que su `RTCPeerConnection` conozca la configuración propuesta por Peer1.
6. Peer2 crea la respuesta SDP llamando al método `createAnswer()`.
7. Peer2 llama al método `setLocalDescription()`.
8. Peer2 utiliza el mecanismo de señalización para hacer llegar la respuesta SDP a Peer1.
9. Peer1 llama al método `setRemoteDescription()` con la respuesta de Peer2.

Para establecer la sesión multimedia, los extremos también necesitan intercambiar información de red. Este proceso se conoce como 'intercambio de candidatos', y se refiere al proceso de encontrar direcciones IP y puertos utilizando ICE. A continuación se describe el proceso acorde con el ejemplo de la Figura 12:

1. Peer1 crea un objeto RTCPeerConnection con el manejador de eventos `onicecandidate`.
2. Este manejador se disparará cuando un candidato de red esté disponible.
3. Peer1 utiliza un mecanismo de señalización para enviar la información del candidato a Peer2.
4. Cuando Peer2 recibe información de un candidato de Peer1, llama al método `addIceCandidate()` para añadir el candidato a la `RTCPeerConnection`.

El resultado final de estos dos procesos es una conexión RTP entre los dos extremos con intercambio de información multimedia en tiempo real.

2.5.1. Canal de Señalización

Como se mencionó anteriormente, la forma en la que se transportan los paquetes para la señalización no está estandarizada, sin embargo, hay varias opciones muy populares a la hora de establecer un canal de señalización.

A continuación, se describen las dos topologías de señalización más populares.

2.5.1.1. Triángulo

Este es el escenario más común actualmente en las aplicaciones WebRTC. En él los navegadores de los participantes de la sesión multimedia están descargando la aplicación del mismo servidor Web.
Se conoce como topología de triángulo por la forma que se produce con la señalización, utilizando el servidor web como intermediario (Figura).

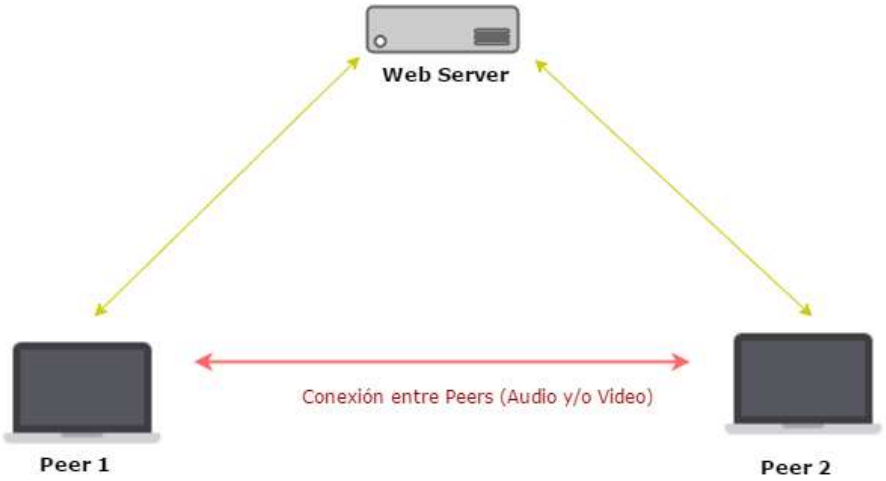

Figura 13: Topología triangular

2.5.1.2. Trapezoide

En este escenario se cuenta con dos servidores web que se comunican entre ellos utilizando algún protocolo de señalización estándar como SIP (Session Initiation Protocol) o Jingle. Esto puede ser de gran utilidad, ya que gran parte de sistemas de comunicaciones VoIP utilizan este protocolo.
Podemos ver la forma de trapezoide en la Figura a continuación:

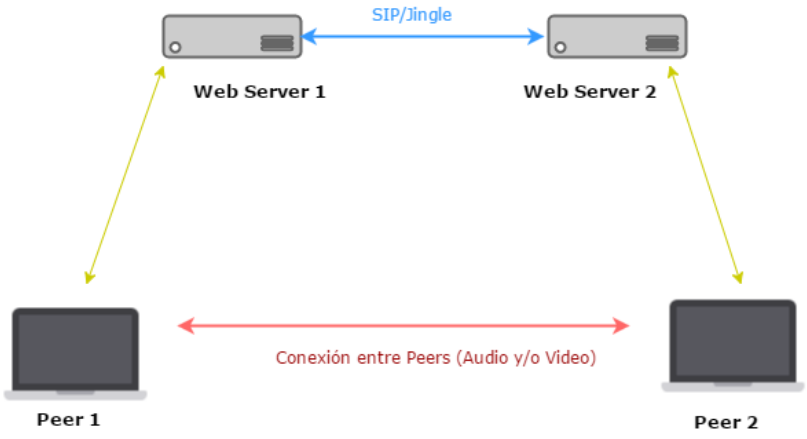

Figura 14: Topología trapezoidal

2.6. MARCO REGULATORIO

Respetando las pautas de la ley orgánica 15/1999, de 13 de diciembre de Protección de datos de carácter personal, (LOPD) [LOPD, 1999] la plataforma multimedia desarrollada para este proyecto no almacenará información de carácter personal de los usuarios, el contenido multimedia compartido por los usuarios será eliminado tras la terminación de la videoconferencia.

El sistema cumple las regulaciones de interés público descritas en la Directiva 2010/13/UE del Parlamento Europeo y del Consejo, de 10 de marzo de 2010, sobre la coordinación de determinadas disposiciones legales, reglamentarias y administrativas de los Estados miembros relativas a la prestación de servicios de comunicación audiovisual (Directiva de servicios de comunicación audiovisual).

3. REQUISITOS DE LA PLATAFORMA

Como se menciona en el capítulo 1, el objetivo principal de este TFG consiste en explorar el potencial ofrecido por la tecnología WebRTC para diseñar, desarrollar y desplegar aplicaciones de comunicación multimedia. Para cumplir este objetivo, se desarrollará una aplicación de conferencia multimedia, que posibilite el intercambio en tiempo real de audio, video y mensajes de texto entre múltiples usuarios.

3.1. DISEÑO Y REQUISITOS FUNCIONALES

La aplicación de videoconferencia seguirá una topología de estrella, similar a la topología de triangulo explicada en el apartado 2.5.1.1. pero con soporte a múltiples extremos. En la figura 15 se muestra el diseño de la aplicación, representándose el intercambio de información entre tres usuarios.

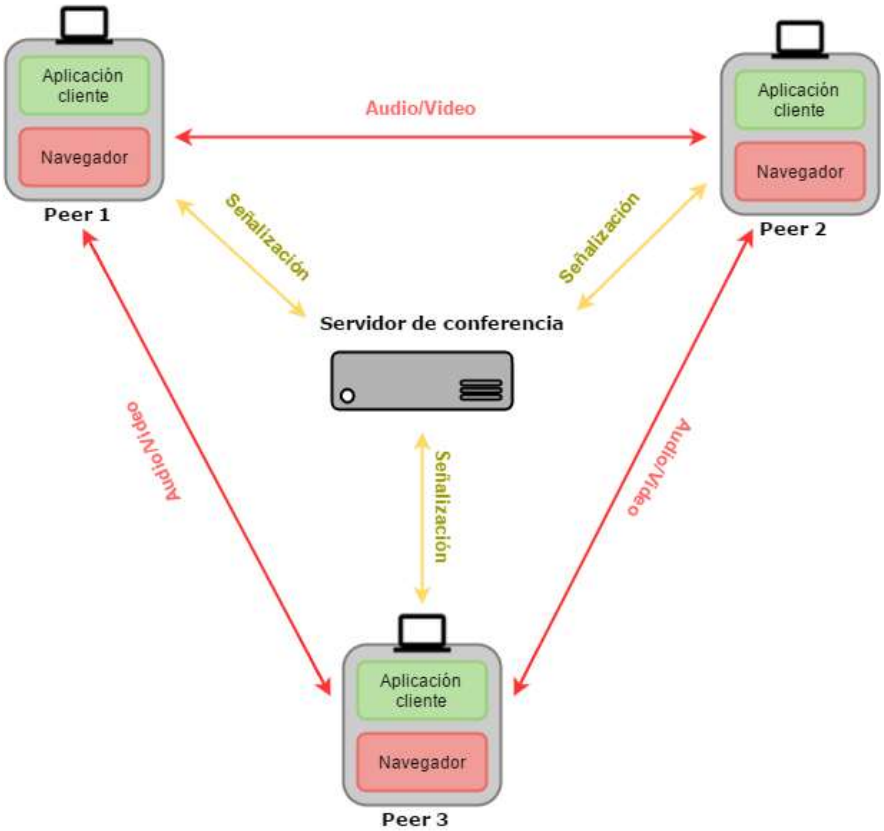

Figura 15: Diseño de la aplicación de conferencia con tres participantes

El sistema contará con una aplicación cliente que se ejecutará en el navegador Web del usuario mediante el soporte ofrecido por WebRTC, que estará habilitado en dicho navegador. El usuario accederá al servicio diseñado a través de un servidor de conferencia, introduciendo una URL en su navegador, lo que le permitirá conectarse a la plataforma. Este servidor intercambiará información de control con la aplicación del cliente, y permitirá establecer una conferencia entre todos los usuarios que accedan a la misma URL. El intercambio de audio y video correspondientes a una conferencia se realizará directamente entre las aplicaciones cliente de los usuarios. A continuación, se detallan los requisitos funcionales que se han establecido para el desarrollo de la aplicación de conferencia:

- Topología de tipo estrella, que es una derivación para múltiples usuarios de la topología triangular (Apartado 2.5.2.1).
- Capacidad para interconectar a múltiples extremos. Cada participante debe enviar y recibir información de todos los demás participantes conectados al servidor de conferencias.
- Soporte de comunicaciones de audio y video entre todos los participantes conectados. Se debe mostrar el audio y video de todos los participantes en la interfaz gráfica de cada usuario en tiempo real.
- Soporte de chat de texto entre los participantes conectados a la llamada.
- Señalización utilizando WebSockets. Cada participante debe establecer una conexión bidireccional con el servidor de conferencia por el que se realizarán los intercambios de mensajes SDP y candidatos ICE (Información de red).

A continuación se describen en mayor detalle los distintos componentes de la aplicación de conferencia, así como sus requisitos de diseño.

3.2. REQUISITOS DEL CLIENTE

En este sub-apartado se describirán los requisitos que debe cumplir la aplicación cliente que funcionará en el Navegador de cada usuario de la plataforma. Los requisitos de diseño son los siguientes:

- Proporcionar una interfaz gráfica simple e intuitiva que permita al usuario acceder al servicio de conferencia. Dicha interfaz debe

mostrar, de manera organizada, el video y los mensajes de texto de múltiples usuarios.
- Capturar los flujos de información multimedia de la cámara y micrófono locales e intercambiar dicho contenido con los demás participantes de la sesión.
- Generar las descripciones de sesión (SDP) de cada usuario y reaccionar utilizando los métodos WebRTC convenientes al recibir mensajes SDP remotos.
- Generar candidatos ICE y reaccionar añadiendo los candidatos ICE remotos que se reciban a la conexión RTC correspondiente.
- Soportar el envío/recepción de mensajes de texto entre los múltiples participantes de la sesión multimedia.

Por otra parte, la aplicación cliente tendrá unos requisitos mínimos de acceso por parte del usuario, que se enumeran a continuación:

- Disponer de una cámara web y un micrófono funcionales
- Conexión a internet
- En principio el desarrollo se hará para el navegador web Google Chrome (versión 48 mínimo), ya que es actualmente el que dispone de más facilidades y herramientas para trabajar con WebRTC.

3.3. REQUISITOS DEL SERVIDOR

En este sub-apartado se describirán los requisitos que debe cumplir el servidor de conferencias. Los requisitos de diseño son los siguientes:

- Llevar un control de presencia de los participantes, es decir, reaccionar con determinados eventos cuando un nuevo participante se conecta, o cuando un participante abandona la aplicación.
- Actuar como intermediario en el intercambio de oferta/respuesta SDP redirigiendo los mensajes al destino correspondiente.
- Actuar como intermediario en el intercambio de candidatos ICE de los participantes.
- Enviar los mensajes de texto que se reciban de cada usuario a todos los participantes conectados.

4. IMPLEMENTACIÓN

En este capítulo se hablará de la forma en la que se ha llevado a cabo la aplicación. Se dividirá en dos bloques: un apartado dedicado a la aplicación cliente, en la que se mostrara la interfaz gráfica de la plataforma, su funcionamiento apoyándonos en varios diagramas y los recursos y herramientas que se han utilizado para su despliegue; y un segundo apartado dedicado al servidor del servicio de conferencia, explicando también en este caso su funcionamiento y el procedimiento para desarrollarlo.

En la figura 16 se muestra una representación de la arquitectura general de la plataforma de conferencia que se ha implementado:

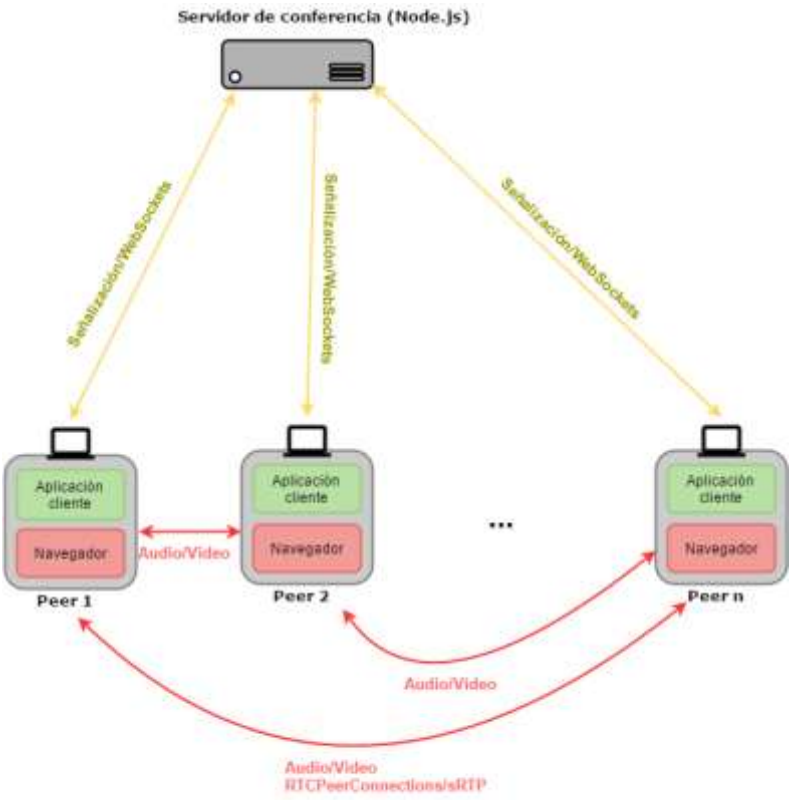

Figura 16: Arquitectura general de la aplicación

El funcionamiento de la aplicación es sencillo, cuando un equipo cliente (o Peer) accede a la página web se abre un socket TCP entre el navegador y el servidor. Este socket hará la función de canal bidireccional de señalización por el que se producirá el intercambio de

mensajes SDP y candidatos ICE entre los participantes de la sesión, utilizando el servidor como intermediario.

Cada vez que se añade un nuevo participante, éste establece un intercambio de mensajes de señalización con los demás participantes conectados a la plataforma, derivando en una conexión multimedia en tiempo real (RTCPeerConnection) con cada uno de ellos.

4.1. APLICACIÓN CLIENTE

En este apartado se hablará del desarrollo y funcionamiento de la aplicación que corre en el navegador de cada participante conectado al servicio de conferencia. Se dedicará un primer sub-apartado a explicar la interfaz gráfica de la aplicación, un segundo sub-apartado a hablar de su funcionamiento, y un tercer sub-apartado a explicar los recursos y herramientas utilizados en el desarrollo de la aplicación.

4.1.1. Interfaz Gráfica

En este sub-apartado se mostrarán las distintas partes de la estructura gráfica de la aplicación, explicando en detalle cada una de ellas. En la Figura 17 se muestra la interfaz gráfica general de la aplicación durante una videoconferencia de tres participantes desde el punto de vista del Cliente 1 (primer usuario en acceder a la aplicación):

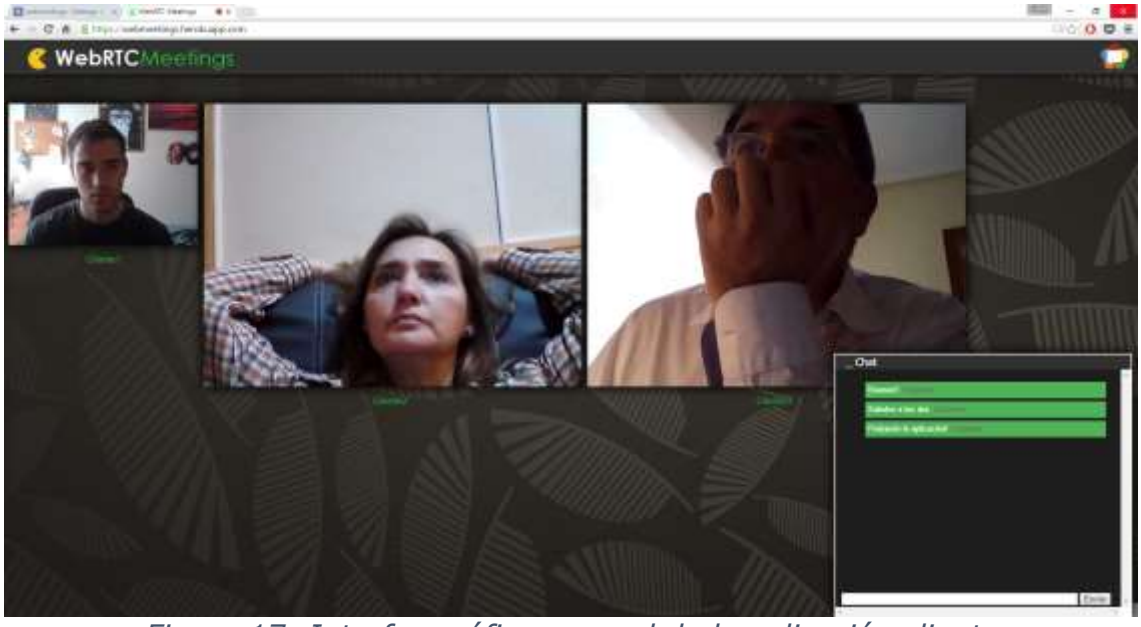

Figura 17: Interfaz gráfica general de la aplicación cliente

En la parte superior derecha, debajo del logo de la aplicación, se encuentra el video local, es decir, la información multimedia de la cámara del cliente que ha accedido a la página web. Debajo del video se muestra una etiqueta que indica el número de cliente por orden de acceso, esta etiqueta hará las funciones de 'nickname'.

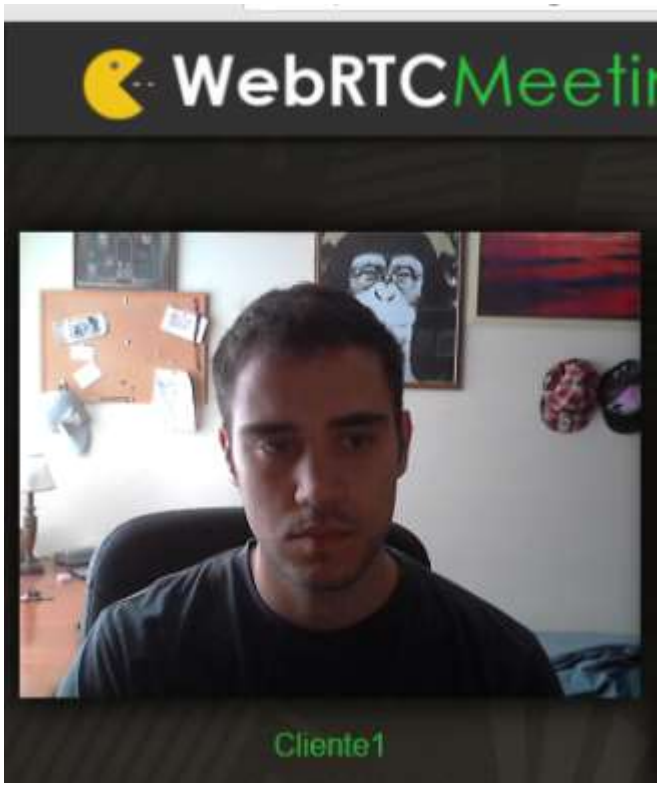

Figura 18: Video local

A medida que van accediendo otros usuarios a la URL asignada a la conferencia, su video aparecerá a la derecha del video local, en un espacio definido como 'video remoto' que tendrá unas dimensiones superiores al video local.

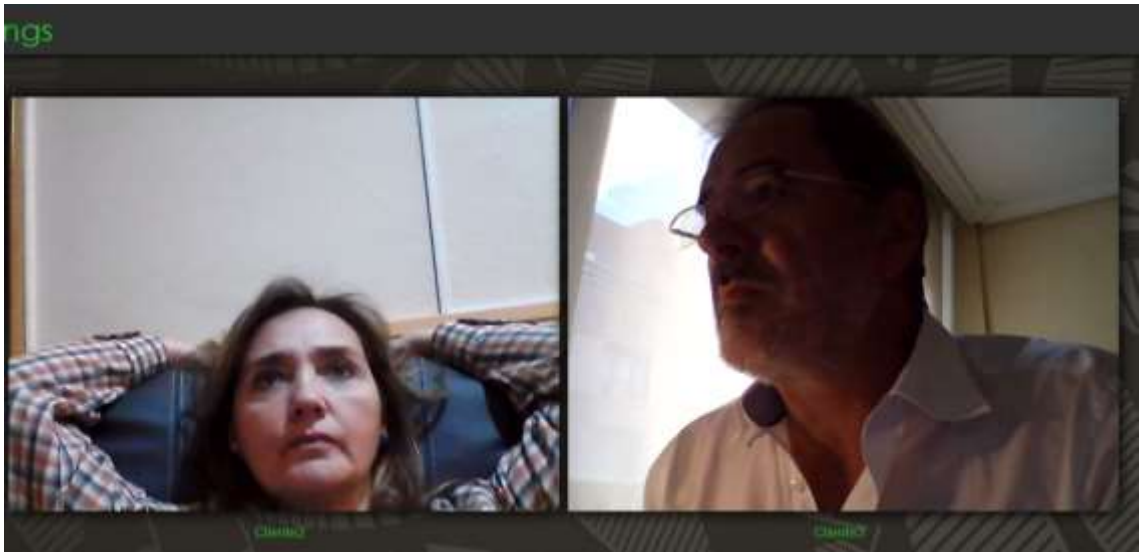

Figura 19: Video remoto

En la esquina inferior izquierda se encuentra la pestaña de chat de texto, que podrá mostrarse u ocultarse cuando el cliente hace click en ella. Los mensajes aparecen dentro de burbujas de color verde y contienen un identificador de color violeta con el identificador de cada cliente que envía un mensaje.

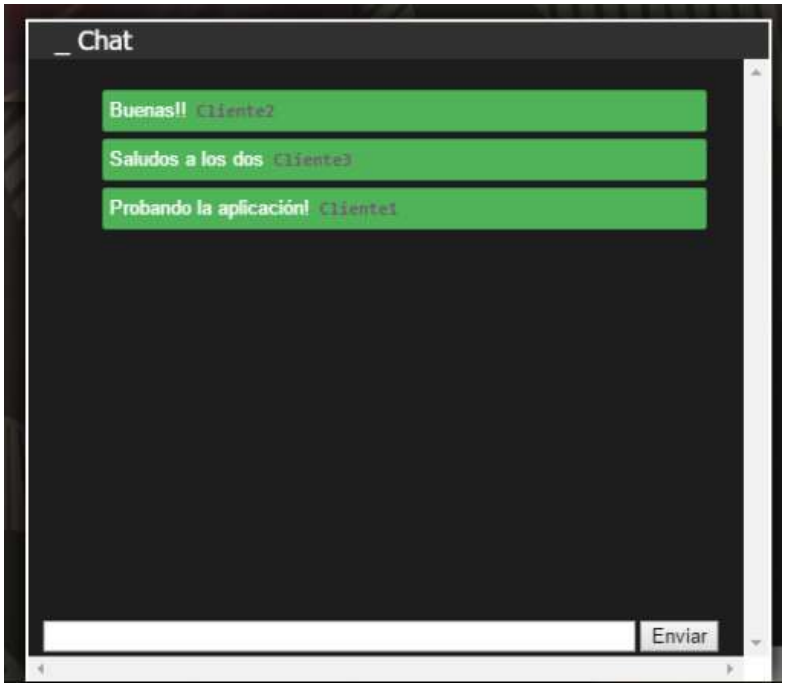

Figura 20: Chat de texto de la aplicación

4.1.2. Recursos y herramientas

Al ser una aplicación WebRTC, los lenguajes de programación utilizados en la aplicación han sido:

- **HTML5** [22]: Para definir la estructura y el contenido de la interfaz gráfica de la aplicación. Los elementos HTML tienen que interactuar con la lógica de la aplicación, para mostrar el contenido multimedia que se obtiene a través de las funcionalidades de WebRTC.

- **CSS3** [23]**:** Para diseñar los estilos y la presentación de la aplicación.

- **JavaScript** [24]**:** Para definir la lógica y el comportamiento de la aplicación, para el desarrollo de ésta aplicación se han utilizado las siguientes herramientas Javascript:

    - **APIs de WebRTC:** se han utilizado dos de las tres APIs principales de WebRTC en su versión más reciente (1.0), RTCPeerConnection y MediaStream (explicadas en detalle en el apartado 2.2) para la configuración de la sesión de videochat.

      No se ha considerado la utilización de la API RTCDataChannel ya que el desarrollo del chat de texto resultaba más sencillo aprovechando el canal de señalización, se habla de este aspecto en más profundidad en el apartado dedicado al servidor de conferencias.

    - **JQuery- (v1.12.3)** [25]**:** Se ha utilizado la biblioteca Jquery para simplificar la manera en la que se interactúa con los elementos HTML desde el código Javascript.

Para escribir y editar el código se ha utilizado el editor de texto **Sublime Text 3**.

4.1.3. Funcionamiento de la aplicación cliente

En la Figura 21 se muestra el diagrama de flujo del funcionamiento de la aplicación cliente para un cliente N posterior al cliente 1. El primer paso cuando un participante accede a la URL desde su navegador es la obtención de la información de la cámara y el micrófono locales (stream local). Posteriormente se crea un objeto RTCPeerConnection y se le añade el stream obtenido. Después de esto se realiza una llamada (envío de oferta SDP) a todos los participantes conectados a la web, por orden descendente.

La aplicación tiene un escuchador de eventos que se dispara cuando se recibe un mensaje del servidor. Según el tipo del mensaje que se reciba la aplicación reaccionará de una forma u otra.

Si el mensaje es de tipo Respuesta SDP, la aplicación llamará al método `setRemoteDescription()` que configura el mensaje recibido como respuesta SDP en la RTCPC (RTCPeerConnection). Tras esto recibirá la información multimedia remota y la asociará al elemento HTML de video correspondiente, pudiendo así ver el video en la interfaz gráfica de la aplicación.

Si el mensaje es de tipo Oferta SDP, la aplicación creará una respuesta SDP y la enviará al servidor para que éste la reenvíe al Cliente que envió la oferta.

Si el mensaje es un candidato ICE (dirección IP y puerto preparados para recibir información multimedia) la aplicación simplemente añadirá esta información a la RTCPC.

Si el mensaje es un mensaje de Chat de texto, la aplicación asociara el contenido del mensaje al elemento HTML correspondiente a una burbuja de chat, pudiendo así ver el mensaje en la interfaz gráfica del programa.

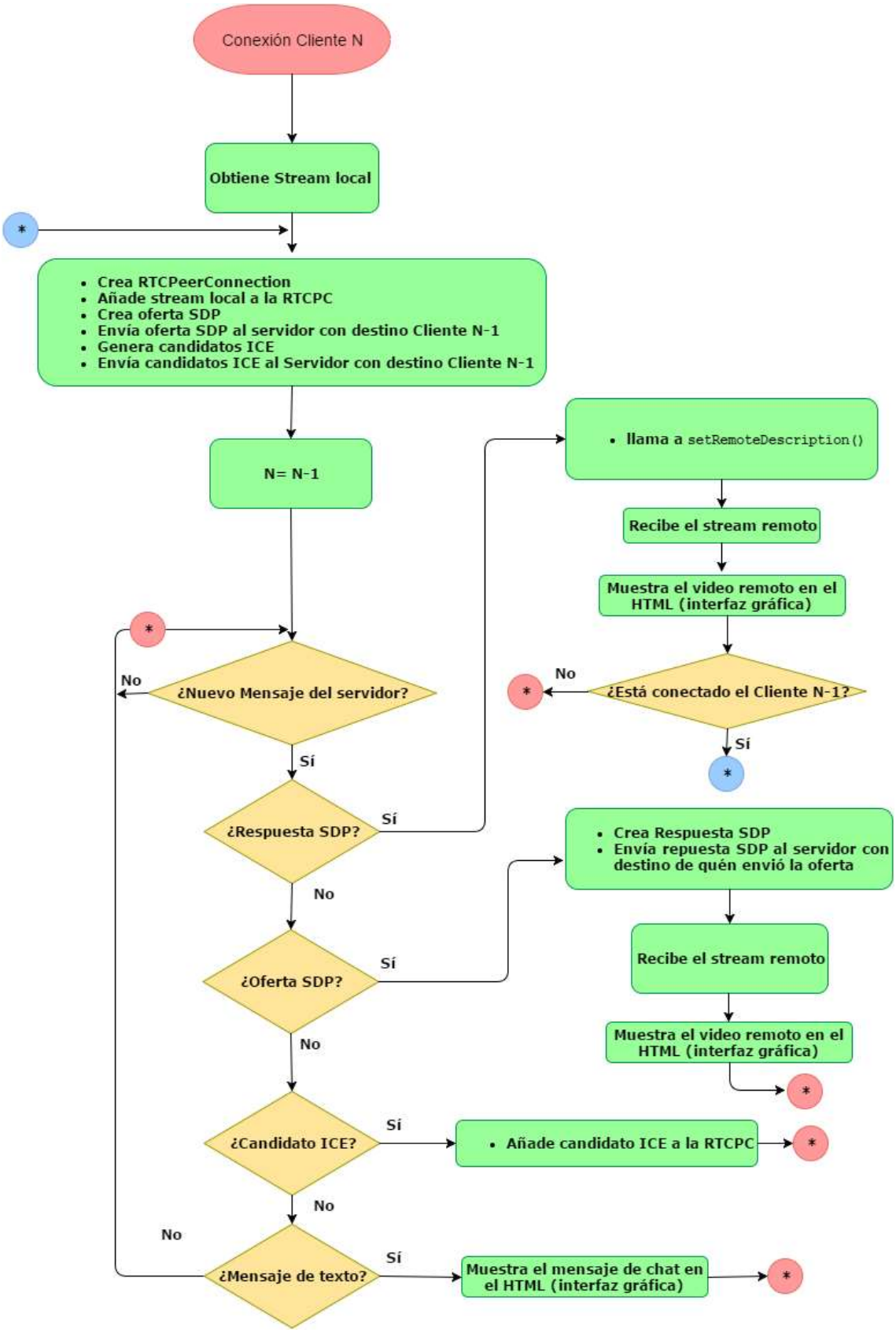

Figura 21: Diagrama de flujo del funcionamiento de la aplicación cliente

4.2. SERVIDOR DEL SERVICIO DE CONFERENCIA

En este caso el servidor que almacena la página web actúa como servidor de conferencia, cuando un cliente abre la URL en su navegador, se abre una conexión TCP bidireccional (socket) entre el navegador y el servidor, que se utiliza para enviar mensajes entre los participantes.

4.2.1. Funcionamiento del servidor de conferencia

El funcionamiento del servidor se basa en enviar y recibir mensajes de la aplicación y disparar determinados eventos en función de los mensajes recibidos. Los mensajes que se envían a través de este servidor pueden ser de los siguientes tipos:

- Mensajes de control: Estos mensajes únicamente se envían entre el usuario y el servidor cuando se disparan determinados eventos de presencia. Cuando un usuario accede a la aplicación se envía un mensaje de tipo 'connect' al servidor para que éste le conceda un identificador y aumente el contador de clientes conectados. A su vez, cuando un participante abandona la aplicación, se envía un mensaje de tipo 'disconnect' a los demás participantes para que la aplicación cliente elimine su espacio de video de la interfaz gráfica.

- Mensajes con carga SDP: Para establecer una videollamada entre dos participantes, el que llama debe enviar su oferta SDP como cadena de texto en un mensaje de tipo 'offer' al servidor (definiendo un destino), éste lo recibirá y lo enviará al destino, produciéndose el intercambio de oferta/respuesta SDP explicado en el apartado 2.5.

- Candidatos ICE: Para llevar a cabo la sesión multimedia también es necesario que se envíen mensajes con los candidatos de red como cadena de texto a través del servidor de conferencia.

- Mensajes de texto: Los mensajes del chat de texto de la aplicación también son mensajes que van a través del socket

navegador-servidor y se envían a todos los participantes conectados a la plataforma. Cuando la aplicación cliente recibe un mensaje de texto del servidor lo muestra en la interfaz gráfica en la pestaña de chat.

Para poder enviar mensajes desde el servidor a un determinado destino, se ha utilizado la funcionalidad de 'salas' de socket.io (más información en el siguiente capítulo). Existen dificultades utilizando esta herramienta para enviar mensajes a un nodo determinado, sin embargo, socket.io permite definir salas que sí pueden usarse como destinatario en el envío de mensajes. Por tanto, para enviar mensajes al destino deseado en la aplicación, cada uno de los participantes se une a una 'sala' diferente al conectarse, que tiene como nombre el mismo nombre de que se utiliza como 'nickname' de cada usuario, es decir, Cliente X (siendo X el número de participante por orden de acceso).

Por ejemplo, cuando se conecta el Cliente 1, se crea una sala con el nombre 'Cliente1' que se puede utilizar como destinatario para enviar mensajes. En la Figura 21 se muestra el comando para enviar un mensaje al Cliente X:

```
io.to("ClienteX").emit('tipo de mensaje', {mensaje});
```

Figura 22: Utilizando salas para enviar mensajes a destinos determinados

En el diagrama de la figura 23 se ejemplifica el funcionamiento de la plataforma completa (aplicación cliente y servidor) para el caso de dos participantes, denominados en la figura "Alice" y "Bob".

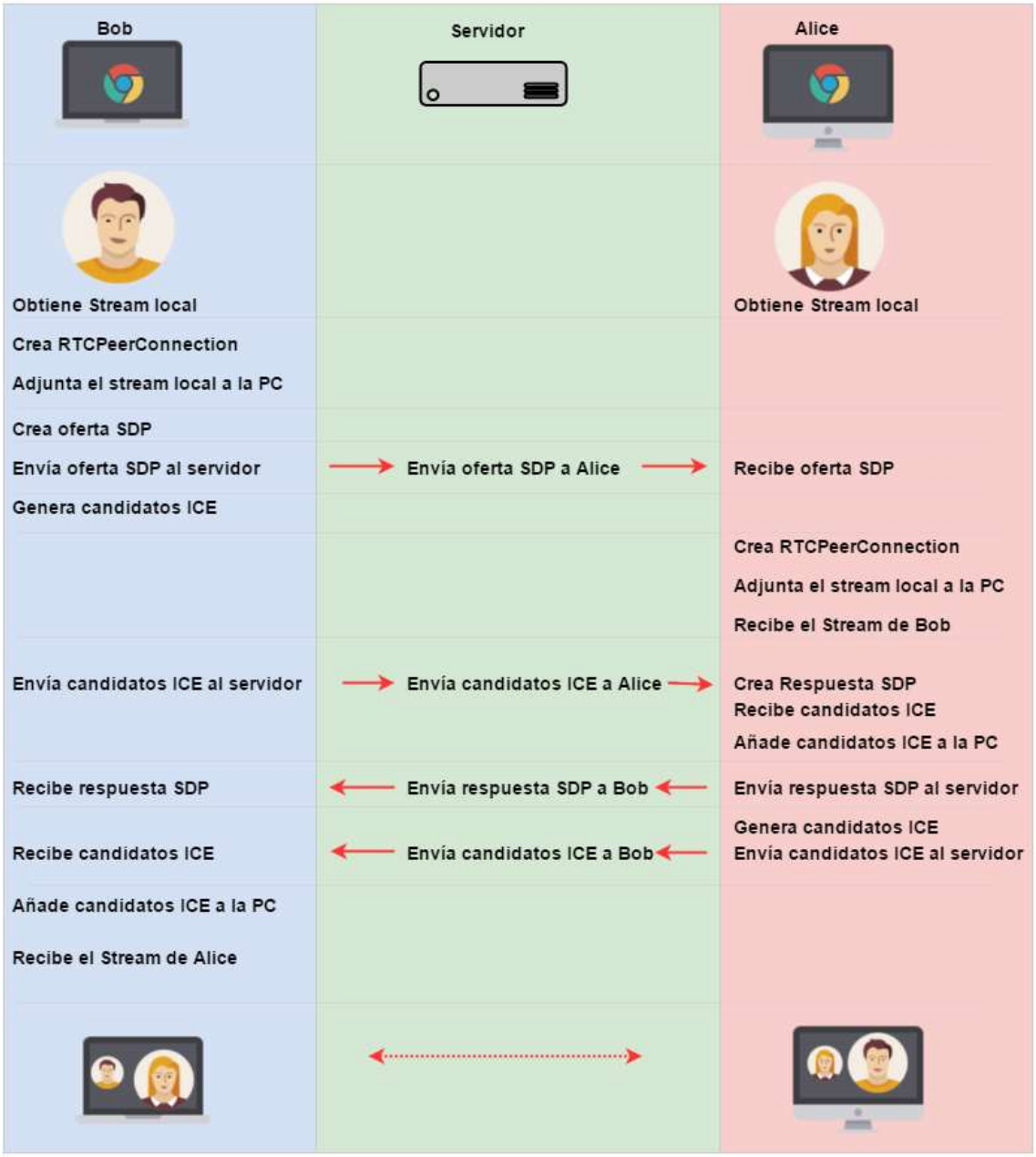

Figura 23: Diagrama de funcionamiento de la aplicación en el caso de dos participantes

4.2.2. Recursos y herramientas para el desarrollo del servidor

El servidor del servicio de conferencia se ha desarrollado en lenguaje JavaScript utilizando las siguientes herramientas:

- **Node.js:** Es un entorno JavaScript que se ejecuta en el lado del servidor, basado en eventos, que utiliza el motor V8 de Google. Cuenta con el ecosistema de paquetes npm, que ofrece una gran cantidad de librerías de código libre, en esta aplicación se han utilizado los siguientes:

    - **Express** [26]**:** Es un framework de desarrollo de aplicaciones web para Node.js
    - **Socket.io:** Es una librería de JavaScript que nos permite crear una conexión bidireccional entre el navegador y el servidor e intercambiar mensajes personalizados que sirvan para disparar determinados eventos. Es una implementación del protocolo WebSocket (apartado 2.3.4) que facilita y proporciona varias funcionalidades, como hacer broadcast a multiples sockets, y entrada/salida asíncrona.
      En el desarrollo de la aplicación se utiliza esta herramienta para realizar las funciones de señalización (intercambio de oferta/respuesta SDP y candidatos ICE) y las funciones de chat de texto grupal.
    - **Adapter.js** [27]**:** Es una librería Javascript que permite desarrollar aplicaciones WebRTC con soporte para varios navegadores utilizando un mismo código.

5. VALIDACIÓN DE LA APLICACIÓN

En este capítulo se describen las pruebas realizadas para verificar el correcto funcionamiento de la plataforma de conferencias desarrollada, así como los resultados de las mismas y una serie de pruebas de rendimiento y consumo en distintos casos de uso. Las pruebas se realizarán en un entorno local con tres ordenadores conectados a una misma red wifi siguiendo la topología mostrada en la Figura 24.

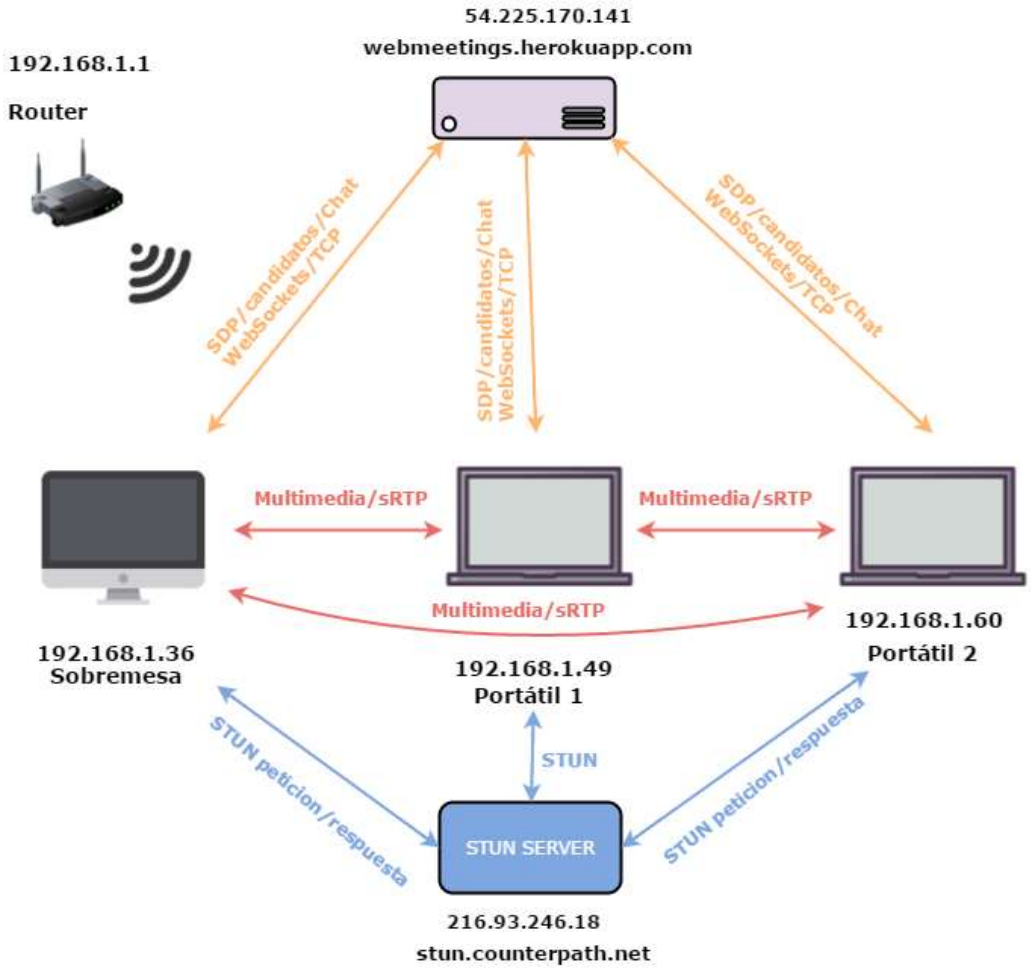

Figura 24: Topología de pruebas

Para llevar a cabo las pruebas de este capítulo, se ha desplegado la aplicación en la plataforma heroku, que es un servicio que permite alojar de forma gratuita (con limitaciones) aplicaciones interactivas basadas en varios lenguajes entre los que se encuentra Node.js.

También se contará con el servidor STUN público stun.counterpath.net, que, aunque no será imprescindible para establecer la conexión WebRTC entre los equipos (están conectados a la misma red local) podremos comprobar y validar la interacción de los equipos con este servidor para obtener su IP pública.

5.1. CHROME WEBRTC INTERNALS

WebRTC Internals es una herramienta del navegador Google Chrome que nos permite comprobar diversas características y test de rendimiento de las aplicaciones WebRTC que estén funcionando en ese momento en el navegador. Accediendo desde Chrome a la URL chrome://webrtc-internals/ con una aplicación WebRTC funcionando en el navegador web local, aparece lo mostrado en la Figura 25:

Figura 25:Pantalla principal de chrome://webrtc-internals

Como se puede ver en la Figura 25, nos aparecen tres pestañas, la primera refleja las peticiones a getUserMedia(), en este caso habrá una única llamada (la que tiene lugar al acceder a la plataforma). Las otras pestañas corresponden a las conexiones RTC activas, en este caso, al ser una llamada de tres participantes hay dos conexiones activas, una por cada participante remoto. En la Figura 26 se puede observar lo que se mostraría en el caso de una llamada con 5 participantes:

Figura 26:Pantalla principal de la herramienta con 5 participantes

Al acceder a una de las pestañas correspondientes a las conexiones activas se muestran dos columnas. La primera es la columna de eventos, en la que se pueden ver los eventos que tienen lugar en esa conexión con su respectiva marca temporal. La segunda columna corresponde a las tablas de estadísticas, donde podemos

comprobar el rendimiento de la aplicación a través de varios gráficos. La visión general de estas dos columnas se muestra en la figura 27.

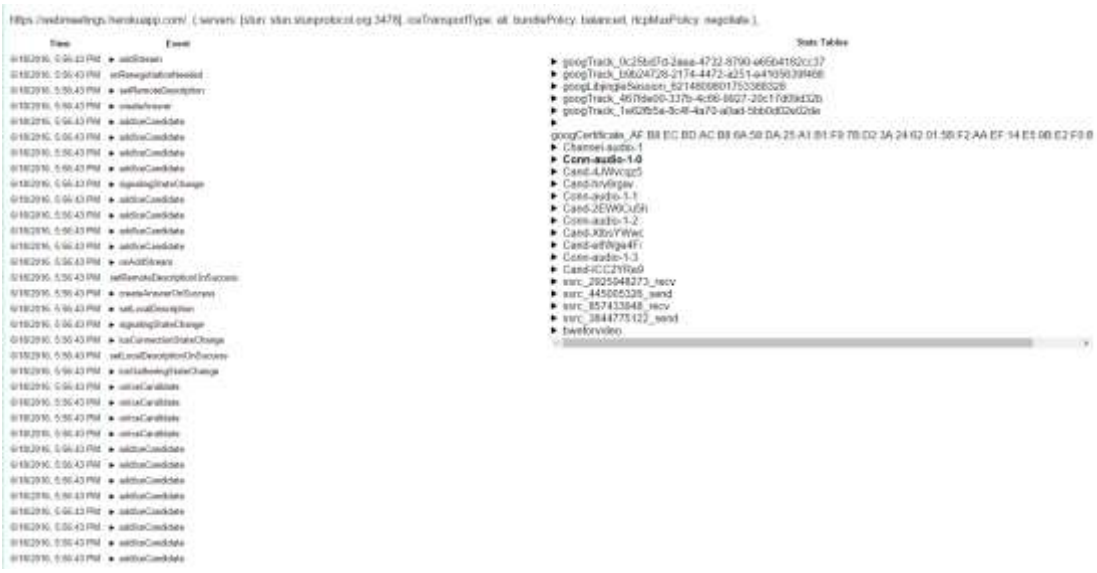

Figura 27: Información que nos ofrece chrome://webrtc-internals

En la columna de eventos, se muestran los mensajes SDP y los candidatos ICE. En las figuras 28 y 29 podemos ver como las direcciones IP de los equipos remotos con lo que se están realizando estas pruebas, son añadidos a su respectiva conexión RTC a través del evento addICECandidate:

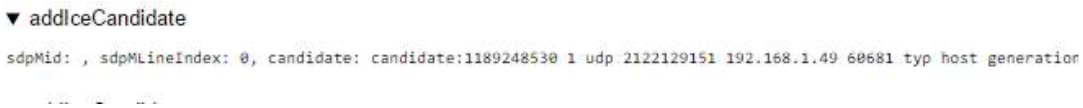

Figura 28: dirección del Portátil 1 añadido como candidato ICE

Figura 29: dirección del Portátil 2 añadido como candidato ICE

También se pueden examinar las descripciones SDP que se intercambian los participantes. En la Figura 30 se muestra un fragmento de la respuesta SDP que envía el PC sobremesa a uno de los portátiles.

```
type: answer, sdp: v=0
o=- 6836261055207001694 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio video
a=msid-semantic: WMS yzjV9bEFDfS48YwtCeRFwl4Y42Bs4dZMwfiF
m=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 126
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=ice-ufrag:oVe7WObFW2l9FEJe
a=ice-pwd:FtbGC3Z4JvJOB5aXGZeBwrCC
a=fingerprint:sha-256 AF:B8:EC:BD:AC:B8:6A:58:DA:25:A1:B1:F9:7B:D2:3A:24:62:01:5B:F2:AA:EF:14:E5:9B:E2:F0:B4:CC:59:
a=setup:active
a=mid:audio
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=sendrecv
a=rtcp-mux
a=rtpmap:111 opus/48000/2
a=rtcp-fb:111 transport-cc
a=fmtp:111 minptime=10;useinbandfec=1
a=rtpmap:103 ISAC/16000
a=rtpmap:104 ISAC/32000
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:106 CN/32000
a=rtpmap:105 CN/16000
a=rtpmap:13 CN/8000
a=rtpmap:126 telephone-event/8000
a=maxptime:60
a=ssrc:1556886826 cname:/WwTxPQ5XAiRN7aa
a=ssrc:1556886826 msid:yzjV9bEFDfS48YwtCeRFwl4Y42Bs4dZMwfiF a4aef7ca-348f-416f-a432-c8c03e69cc3a
a=ssrc:1556886826 mslabel:yzjV9bEFDfS48YwtCeRFwl4Y42Bs4dZMwfiF
a=ssrc:1556886826 label:a4aef7ca-348f-416f-a432-c8c03e69cc3a
m=video 9 UDP/TLS/RTP/SAVPF 100 101 116 117 96 97 98
```

Figura 30: Fragmento de Respuesta SDP entre dos participantes

A continuación se van a analizar algunas de las líneas de este fragmento de respuesta SDP:

o=- 6836261055207001694 2 IN IP4 127.0.0.1

El primer número es un identificador único de la sesión. El segundo número es la versión de la sesión, si otro mensaje oferta/respuesta SDP es requerido en esta sesión este número incrementara en uno. Los siguientes tres números son el tipo de red (internet), la versión de la dirección IP, y la dirección IP del equipo que creó el mensaje SDP.

a= group: BUNDLE audio video

El agrupamiento BUNDLE establece una relación entre varias líneas incluidas en el SDP, comúnmente audio y video. En WebRTC existe una multiplexación [28] de varios flujos multimedia. En este caso se acepta la multiplexación de audio y video.

a = msid-semantic: WMS yzjv9bEFDfS48YwtCeRFw14Y42Bs4dZMwfiF

Esta línea da un identificador único para el flujo multimedia de WebRTC (WMS, WebRTC Media Stream).

```
m = audio 9 UDP/TLS/RTP/SAPF 111 103 104 9 0 8 106 105 13 126
```

Las líneas m son las líneas de información multimedia. El primer elemento es el tipo, en este caso audio. El tercer elemento son los protocolos de transporte que van a usarse para la sesión, y los números del final son los formatos multimedia que soporta el navegador para enviar y recibir información multimedia.

```
c= IN IP4 0.0.0.0
```

Las líneas c son las líneas de conexión. La dirección IP que se muestra es la dirección en la que se desea recibir tráfico multimedia en tiempo real. Como en WebRTC ICE prevalece para especificar la información de red (candidatos) esta línea no se utiliza.

```
a = rtpmap: XXXXX
```

Estas líneas especifican los códecs a utilizar en la sesión multimedia.

A continuación se muestran algunos gráficos que se incluyen en la columna de tablas de estadísticas de la herramienta webrtc-internals. En las Figuras 31 y 32 podemos ver las estadísticas de los paquetes multimedia correspondientes al audio que se está transmitiendo y recibiendo.

Figura 31: Gráfica de audio enviado

Figura 32: Gráfica de audio recibido

En las figuras 33 y 34 podemos ver las estadísticas equivalentes a los paquetes de video.

Figura 33: Gráfica de video enviado

Figura 34: Gráfica de video recibido

Como se puede observar no se están perdiendo paquetes en esta sesión.

5.2. WIRESHARK

En este apartado se van a realizar pruebas utilizando la herramienta Wireshark para capturar y analizar el tráfico de red de la aplicación.

En la figura 35 se puede ver una captura de Wireshark correspondiente al intercambio de paquetes entre el equipo y el servidor de conferencias. Como se explicó anteriormente para realizar esta conexión utilizamos la herramienta socket.io, esto se traduce en un canal TCP bidireccional entre el servidor y el cliente (canal de señalización de la aplicación). Por medio de este canal se intercambian los mensajes SDP, los candidatos ICE, y los mensajes del chat de texto de la aplicación.

```
54.225.170.141    192.168.1.36      TCP    60 443 → 52736 [ACK] Seq=1 Ack=453 Win=97 Len=0
54.225.170.141    192.168.1.36      TCP    60 443 → 52737 [ACK] Seq=1 Ack=505 Win=87 Len=0
54.225.170.141    192.168.1.36      TCP    60 443 → 52733 [ACK] Seq=1 Ack=485 Win=83 Len=0
192.168.1.36      54.225.170.141    TCP    54 52735 → 443 [ACK] Seq=472 Ack=324 Win=259 Len=0
192.168.1.36      54.225.170.141    TCP    54 52736 → 443 [ACK] Seq=453 Ack=171 Win=260 Len=0
192.168.1.36      54.225.170.141    TCP    54 52738 → 443 [ACK] Seq=491 Ack=325 Win=259 Len=0
```

Figura 35: Mensajes entre el cliente y el servidor de conferencia

En la figura 36 se pueden comprobar los mensajes enviados al servidor STUN que se está utilizando en esta prueba (stun.counterpath.net, 216.93.246.18). Se puede observar que los primeros dos mensajes son una petición al servidor STUN para obtener la IP pública del equipo, y los siguientes dos mensajes son la respuesta del servidor con la dirección solicitada (tapada con rectángulos negros por motivos de seguridad).

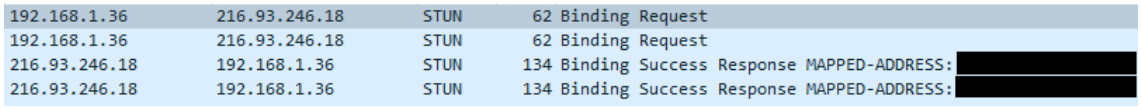

Figura 36: Comunicación con el servidor STUN

Cuando el intercambio de ofertas y respuestas SDP se ha completado con éxito, y se han intercambiado los candidatos ICE, comienza la sesión RTP de audio y video entre los tres equipos. El intercambio de paquetes RTP entre los equipos se muestra en la figura 37.

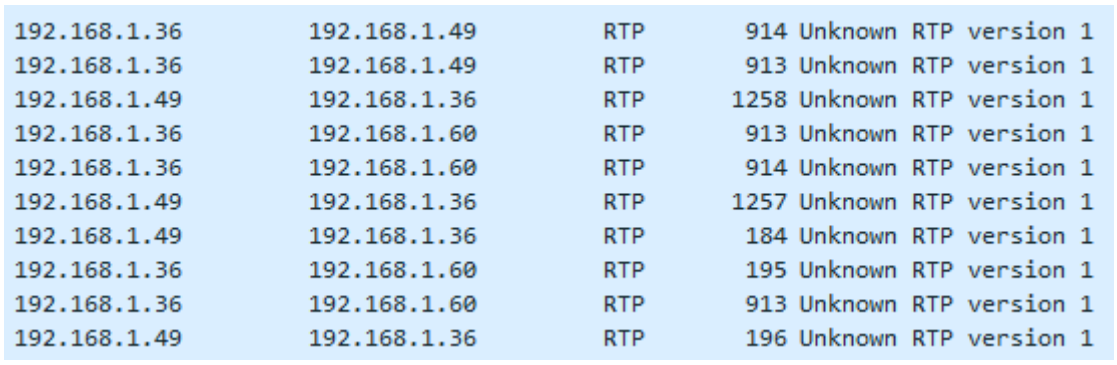

Figura 37: Tráfico RTP entre los participantes de la sesión (Fragmento)

5.3. CONSUMO DE RECURSOS DE LA APLICACIÓN

En este sub-apartado se pretende probar el consumo de recursos de las aplicaciones WebRTC en los tres equipos con los que se están realizando estos test de funcionamiento. Las especificaciones de los equipos son las siguientes:

- Sobremesa
  - Procesador: AMD Six-Core Processor 3.50 Ghz
  - Memoria Ram: 8 Gb

- Portátil 1:
  - Procesador: Intel Core i5 2.70 Ghz
  - Memoria Ram: 4 Gb

- Portátil 2:
  - Procesador: Intel Celeron 2.16 Ghz
  - Memoria Ram: 4Gb

Con una conferencia activa entre los tres equipos y sin ninguna aplicación adicional funcionando se muestran en las Figuras 38, 39 y 40 los datos de consumo.

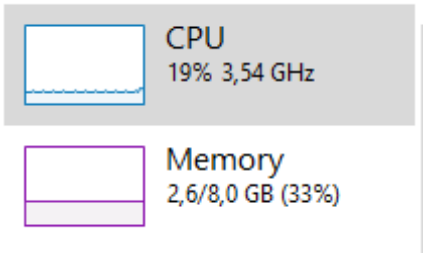

Figura 38: Consumo Sobremesa

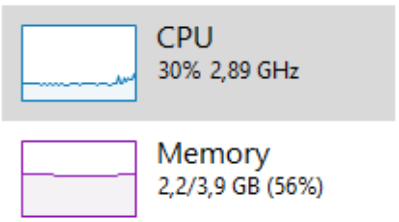

Figura 39: Consumo Portátil 1

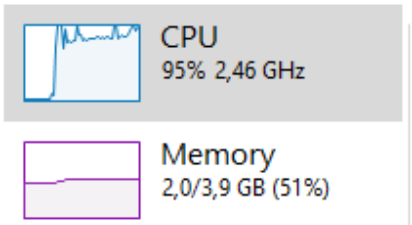

Figura 40: Consumo Portátil 2

Como se puede ver en la Figura 40, el equipo menos potente (portátil 2), utiliza casi el 100% de CPU en la conferencia, lo que se refleja en un rendimiento insuficiente en la aplicación. Podemos deducir de esto que a la tecnología WebRTC todavía le falta optimización para equipos poco potentes.

La siguiente prueba se realizará en el equipo más potente (Sobremesa), se mostrará la información de consumo de recursos a medida que se va aumentando el número de participantes de la conferencia, abriendo varias pestañas de la aplicación en ese mismo equipo.

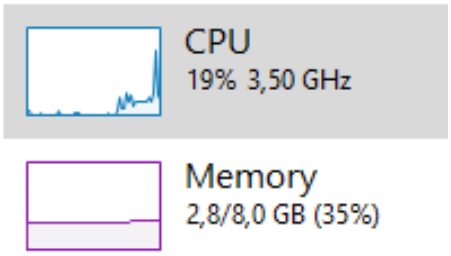

Figura 41: Consumo con dos participantes

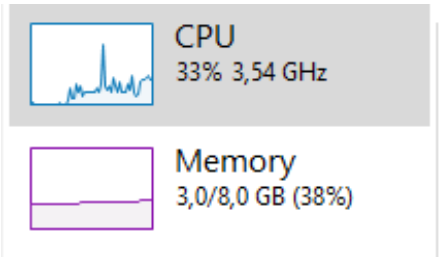

Figura 42: Consumo con tres participantes

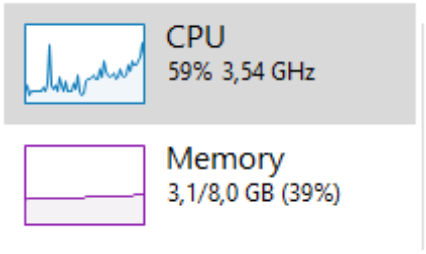

Figura 43: Consumo con cuatro participantes

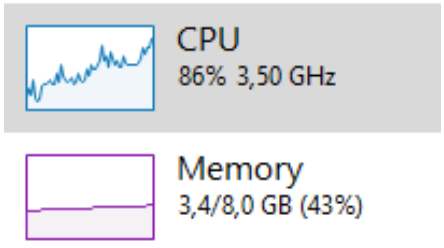

Figura 44: Consumo con cinco participantes

Como se puede observar en las Figuras 41-44, el consumo de la aplicación aumenta aproximadamente un 10-20% cuando se añade un nuevo participante. A partir de 4 participantes el rendimiento de la aplicación se reduce considerablemente, y es a partir del quinto participante, cuando se considera que la aplicación deja de ser funcional.

6. CONCLUSIONES Y LINEAS FUTURAS

En este capítulo se llevará a cabo una valoración final del proyecto. Se presentarán las conclusiones sobre la tecnología WebRTC, valorando su utilidad y madurez, y se hablará de los obstáculos y facilidades que se han encontrado al desarrollar una plataforma basada en esta tecnología. También se hablará sobre los casos de uso potenciales de WebRTC en general y de las líneas futuras de la aplicación desarrollada en particular.

6.1. MADUREZ DE LA TECNOLOGÍA WEBRTC

El desarrollo y despliegue de una plataforma basada en WebRTC, así como la investigación y conocimiento de sus capacidades, ha evidenciado el gran potencial de esta tecnología en un futuro próximo.

La posibilidad de cualquier desarrollador de elaborar un sistema de comunicaciones en tiempo real, unido al crecimiento exponencial de dispositivos conectados a la red, puede suponer una revolución en los medios de comunicación tradicionales y un cambio radical de la forma en la que percibimos la web.

Esta nueva percepción de la web podría traer innovaciones en la metodología tradicional de los servicios, como consultas de medicina virtuales en tiempo real, o formación académica a distancia interactuando en tiempo real con el profesor.

Sin embargo, para que esto suceda, aún hace falta perfeccionar ciertos aspectos de WebRTC, entre los que destacaría el soporte para la mayoría de plataformas, un aumento significativo de los codecs disponibles, y una documentación más clara y accesible.

6.2. OBSTACULOS Y FACILIDADES EN EL DESARROLLO DE LA PLATAFORMA

Como era esperado, se han cumplido los objetivos planteados al inicio del proyecto. Se ha trabajado con las herramientas de código libre ofrecidas por Google, desarrollando y desplegando una plataforma

funcional de videoconferencias en tiempo real con soporte para múltiples usuarios.

Aunque las herramientas para trabajar con WebRTC son accesibles para cualquiera, la documentación y las ayudas para aprender a desarrollar aplicaciones basadas en esta tecnología no son demasiado abundantes y en ocasiones pueden resultar confusas. Esto también se debe a que la tecnología no está asentada aún y se cambian capacidades y funciones bastante a menudo.

Otra limitación importante a la hora de ser desarrollador WebRTC es que, si bien hay soporte para diversos navegadores (tabla de compatibilidad en apartado 2.1.2.4.) resulta bastante complicado crear un producto compatible con varios a la vez, siendo Google Chrome la plataforma más asequible en este aspecto bastante alejada de los demás en facilidad y herramientas de desarrollo.

También se puede considerar una dificultad el corto abanico de opciones para alojar una aplicación web de este tipo sin utilizar servidores externos, ya que se requiere que el servidor web interactúe con los navegadores de los clientes y haga las funciones de servidor de señalización. Existen opciones como heroku que permiten llevar a cabo esta tarea, pero sus limitaciones hacen que las aplicaciones interactivas tengan problemas de estabilidad y su funcionamiento no sea completamente sólido y fiable.

6.3. LINEAS FUTURAS

Aprovechando las características que nos ofrece WebRTC, en un futuro se podrían añadir las siguientes funcionalidades y mejoras a la aplicación:

- Sistema de transferencia de archivos utilizando la API RTCDatachannels, para que, mediante unos ajustes en la interfaz gráfica, los participantes de la conferencia puedan compartir archivos en tiempo real.
- Compartición de pantalla de los participantes de la conferencia. Se exploró su viabilidad, pero la documentación encontrada sobre esta mejora resultó no ser funcional sin complementos de terceros.
- Mejoras de seguridad, como la necesidad de identificarse utilizando un nombre de usuario y una contraseña para acceder a la conferencia. Aprovechando esta información de acceso,

mostrar en la interfaz una etiqueta con el nombre de usuario en el video correspondiente.

Estas mejoras permitirán además acometer la exploración del potencial de WebRTC para proporcionar otros servicios multimedia distintos de los considerados en este TFG, completando así el trabajo de análisis iniciado en este proyecto.

REFERENCIAS

[1] I. Hickson, «WebRTC 1.0: Real-time Communication Between Browsers,» p. https://www.w3.org/TR/webrtc/#references.

[2] J. Valin, «Definition of the Opus Audio Codec,» p. https://tools.ietf.org/html/rfc6716, September 2012.

[3] T. l. G. P. P.Huart, «RTP PAyload Format for tha iSAC Codec,» pp. https://tools.ietf.org/id/draft-ietf-avt-rtp-isac-02.html#isac, October 2012.

[4] S. Andersen, «Internet Low Bit Rate Codec,» p. https://www.ietf.org/rfc/rfc3951.txt, Diciembre 2004.

[5] J. J.Koleszar, «VP8 Data Format and Decoding Guide,» p. https://datatracker.ietf.org/doc/rfc6386/, Noviembre 2011.

[6] I.-T. R. H.264, «Advanced video coding for generic audiovisual services,» Marzo 2010.

[7] A. Grange, «A VP9 Bitstream Overview,» pp. https://tools.ietf.org/html/draft-grange-vp9-bitstream-00, Febrero 2013.

[8] I. H. G. Inc, «Media Capture and Streams,» pp. https://www.w3.org/TR/mediacapture-streams/, Ultima edición Mayo 2016.

[9] R.Jesup, « WebRTC Data Channels,» pp. https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-13, Enero 2015.

[10] R. U. I. T. I. Soociety, «Hypertext Transfer Protocol -- HTTP/1.1,» p. https://tools.ietf.org/html/rfc2616, Junio 1999.

[11] H. S. C. University, «RTP: A Transport Protocol for Real-Time Applications,» p. https://tools.ietf.org/html/rfc3550, Julio 2003.

[12] D. M. M. Baugher, « The Secure Real-time Transport Protocol (SRTP),» p. https://tools.ietf.org/html/rfc3711, Marzo 2004.

[13] M. Handley, «SDP: Session Description Protocol,» p. https://tools.ietf.org/html/rfc4566, Julio 2006.

[14] I. Fette, «The WebSocket Protocol,» p. https://tools.ietf.org/html/rfc6455, Diciembre 2011.

[15] «NodeJS main site,» p. https://nodejs.org/en/.

[16] «Socket IO main site,» p. http://socket.io/.

[17] R. S. M.Ramalho, «Stream Control Transmission Protocol (SCTP),» p. https://tools.ietf.org/html/rfc3758, Mayo 2004.

[18] J. Rosenberg, « Interactive Connectivity Establishment (ICE):A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,» p. https://tools.ietf.org/html/rfc5245, Abril 2010.

[19] J. Rosenberg, «Session Traversal Utilities for NAT (STUN),» p. https://tools.ietf.org/html/rfc5389, Octubre 2008.

[20] R.Mahy, «Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN),» p. https://tools.ietf.org/html/rfc5766, Abril 2010.

[21] U. Jennings, «JavaScript Session Establishment Protocol,» pp. https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03#section-1.1, Ultima edicion Marzo 2016.

[22] I. Hickson, «A vocabulary and associated APIs for HTML and XHTML,» p. https://www.w3.org/TR/html5/, Febrero 2004.

[23] T. A. Jr, «CSS Snapshot 2015,» p. https://www.w3.org/TR/CSS/, Octubre 2015.

[24] w3c, «JAVASCRIPT WEB APIS,» p. https://www.w3.org/standards/webdesign/script.

[25] T. j. Foundation, «Jquery,» [En línea]. Available: https://jquery.com/.

[26] StrongLoop, «Express JS main page,» [En línea]. Available: http://expressjs.com/es/.

[27] K. Jansson, «WebRTC Adapter,» [En línea]. Available: https://github.com/webrtc/adapter.

[28] C. Holmberg, « Negotiating Media Multiplexing Using the Session Description Protocol (SDP) draft-ietf-mmusic-sdp-bundle-negotiation-30.txt,» pp. https://tools.ietf.org/html/draft-ietf-mmusic-sdp-bundle-negotiation-30, Última vez editado 7 de Junio de 2016.

[30] D. C. B. Alan B. Johnson, WebRTC: APIs and RTCWeb Protocols of the HTML5 Real-Time Web.

[31] R. Manson, Getting Started with WebRTC.

[32] Google, «WebRTC.org,» [En línea]. Available: webrtc.org.

ANEXO A. PLANIFICACIÓN Y PRESUPUESTO DEL PROYECTO

En este capítulo se describe la organización que se ha seguido en el desarrollo del proyecto, así como el tiempo empleado y un presupuesto estimado de los recursos utilizados.

A.1. PLANIFICACIÓN DEL PROYECTO

En la Figura 45 se pueden ver las distintas tareas que han tenido lugar en el desarrollo de este proyecto junto a las fechas de inicio y fin de las mismas y su duración en días.
La Planificación del proyecto se ha dividido en tres fases principales:

- **Planificación y organización:** Esta primera fase contiene las tareas previas al desarrollo; el tiempo dedicado a la investigación, el estudio de la tecnología, y el planteamiento del problema junto con la organización del trabajo futuro.

- **Desarrollo:** La fase de desarrollo contempla el proceso de creación de la plataforma y la depuración y pruebas de la misma.

- **Cierre y documentación:** Contiene los procesos de redacción de la memoria y preparación y elaboración de la presentación para la defensa.

Se ha estimado una media de 3 horas de dedicación por día trabajado.

Nombre	Fecha de inicio	Fecha de fin	Duración
Planificacion y organizacion	13/10/15	20/01/16	93
Investigación	13/10/15	29/10/15	17
Estudio de WebRTC	2/11/15	7/12/15	35
Lectura y comprension de la documentación	25/11/15	9/12/15	13
Análisis previo de viabilidad	11/12/15	18/12/15	8
Estudio y elección de las herramientas necesarias	21/12/15	14/01/16	21
Definición del Plan de trabajo	15/01/16	20/01/16	6
Desarrollo	22/01/16	10/05/16	107
Pruebas de los metodos básicos	22/01/16	5/02/16	15
Desarrollo de la aplicación	6/02/16	15/04/16	68
Perfeccionamiento de la interfaz gráfica	18/04/16	25/04/16	8
Despliegue	14/04/16	21/04/16	8
Pruebas	22/04/16	29/04/16	8
Corrección de bugs y errores	2/05/16	10/05/16	9
Cierre y documentación	6/05/16	2/07/16	58
Redacción de la memoria	6/05/16	21/06/16	47
Elaboración de la presentación	22/06/16	2/07/16	11

Figura 45: Desglose de tareas en el desarrollo del proyecto

En la figura 46 a continuación, se muestra el diagrama de Gantt de las tareas descritas anteriormente.

Figura 46:Diagrama de Gantt del proyecto

A.2. PRESUPUESTO

Se pretende dedicar este apartado a realizar una estimación del coste total del proyecto, considerando los recursos de hardware y software utilizados, así como el coste del personal.

El coste de los recursos materiales se compone de equipo utilizado para la realización del proyecto y el coste de las licencias necesarias para el uso del software utilizado. Se muestra el cálculo del coste total de los recursos materiales en la Figura 47.

Descripción	Coste	% dedicado uso proyecto	Dedicación (meses)	Periodo de depreciación (meses)	Coste imputable
PC de sobremesa	700 €	100 %	9	60	105 €
Ordenador Portátil 1	500 €	100 %	9	60	75 €
Ordenador Portátil 2	250 €	100 %	9	60	37,5 €
WebCam	30 €	100 %	9	60	4,5 €
Sublime text 3	0 €	100 %	9	60	0 €
Windows 10	0 €	100 %	9	60	0 €
Microsoft Office	0 €	100 %	9	60	0 €
Navegador Google Chrome	0 €	100 %	9	60	0 €
Heroku Cloud Application Platform	0 €	100 %	9	60	0 €
Altavoces	50 €	100 %	9	60	7,5€
GantProject	0 €	100 %	9	60	0 €
					229,5 €

Figura 47: Presupuesto de los recursos materiales[5]

La amortización se ha calculado mediante la fórmula de la Figura 48.

$$\frac{A}{B} * C * D$$

A: Tiempo (meses) desde la fecha de facturación.

B: Periodo de depreciación.

C: Coste del equipo sin IVA.

D: % de uso dedicado al proyecto.

Figura 48: Fórmula para el cálculo de la amortización

El coste del personal se calcula en la figura 49. Se han tenido en cuenta los datos de la planificación detallados en la Figura 45 con una jornada laboral de 3 horas diarias de media de lunes a viernes. También se ha incluido las horas de tutoría tal y como aparece en la Normativa general para trabajos de Fin de Grado.

[5] Los recursos con coste 0 € corresponden a programas open-source o software ofrecido de forma gratuita por la Universidad Carlos III de Madrid

Nombre	Categoría	Dedicación (horas)	Coste/hora	Coste imputado
Federico Martín Sánchez	Ingeniero	546	20	10.920 €
Iván Vidal Fernández	Ingeniero Senior	60	35	2.100 €
				13.020 €

Figura 49 Coste total del personal

Por último, en la Figura 49 se muestra el cálculo del coste total, incluyendo una tasa de costes indirectos del 20 %.

Descripción	Coste imputado
Amortización	229,5 €
Coste del personal	13.020 €
Costes indirectos	2.649,9 €
	15.899,4 €

Figura 50: Coste total del proyecto

ANEXO B. INTRODUCTION

B.1. MOTIVATION

Is hard to realize how quick technology changes. 20 years ago mobile phones were a revolution and having an internet connection at home was not that common. Nowadays, most of the people have several Internet connected devices, which has resulted a constant evolution in telecommunications, that are gradually moving from traditional phone calls to be mostly through the Internet.
In May 2011, Google presented an open source project known as WebRTC [1], which pretended to introduce real-time communications through the browser and change our perception of the web. The fact that we could connect browsers and share audio and video streaming mean a huge innovation regarding the telephony and conference industry. But this is just the beginning, the ability for JavaScript developers to be able to work with the user local camera and microphone stream, or other user available user media (ie. Screen sharing or local files), opens a new range of possibilities and creates a more dynamic web that allow applications interact with the user in real time through voice, gestures, and a lot new options.

Until now we could get something like these functionalities in the browser with the help of plugins like flash, but this brought problems like crashes or constant updates, which resulted in a low quality perception in the eyes of the consumer.
On the other hand, it is hard to talk about video conferences without thinking about Skype, *What advantages can WebRTC present against Skype?*

In the first place, Skype is an application, it's a program that requires an installation in a machine and offers limited features. However, WebRTC is a set of tools available to everyone, that allows web developers to take advantage of their web programming knowledge (HTML, CSS and JavaScript) to add real time media communications to their creations.
This brings us two huge advantages:

- WebRTC based applications doesn't require of any plugin, just access the site though the browser.
- The fact that is an open source technology, unleashes the creativity of the developers to bring new ideas. Applications

based in this technology may mean a step forward in fields from gaming to e-learning or online streaming in the browser.

According to the arguments above, the motivation for this thesis has been to explore the potential of WebRTC technology to support the development of multimedia communication applications. The following section describes this objective in detail, as all sub-objectives that come from it.

B.2. OBJECTIVES

The main objective for this bachelor thesis is to analyze, from a practical perspective, the support brought by WebRTC technology to design, develop and deploy multimedia communication applications. To do this, the development of a multiparty conference application, that support exchange of audio, video, and text messages between the participants.

In line with this main objective, the following sub-objectives are defined:

- Analyze the state of art, reviewing its technical updated information about WebRTC operation, architecture, protocols, standards and tools, such as the most popular implementations.
- Assess the real possibilities to develop a WebRTC based product, considering difficulties, obstacles and facilities that we face in the current scenario.
- Explore the potential use cases of this technology.
- Reach a solid conclusion and make an assessment of the usefulness and maturity of the WebRTC technology

B.3. STRUCTURE OF THIS DOCUMENT

The contents of each chapter of this thesis are briefly described above:

Chapter 1. Introduction: In the present chapter, we introduce the motivations that lead to build this work, such as its objectives and the structure of the document, which contain a brief description of each chapter.

Chapter 2. State of art of WebRTC: In this Chapter we explain how WebRTC works from a technical perspective, discovering its current limitations and benefits, such as the most popular architectures.

Chapter 3. Platform requirements: This chapter describes the developed application functional requirements and the required aspects for the user to run it.

Chapter 4. Implementation: Splitting the platform in two blocks: Client Application and conference server, this chapter details the design and operation of the application, with help of screenshots and flowcharts. The resources and tools to develop the platform are also described in this chapter.

Chapter 5. Testing: this chapter is meant to show the results of several application tests, showing use tests and checking the application performance.

Chapter 6: Conclusions: This last section is intended to make a general assessment of the technology, discuss its maturity and usefulness in a near future.

ANEXO C. ENGLISH EXTENDED SUMMARY

C.1. INTRODUCTION AND MOTIVATION

Is hard to realize how quick technology changes. 20 years ago mobile phones were a revolution and having an internet connection at home was not that common. Nowadays, most of the people have several Internet connected devices, which has resulted a constant evolution in telecommunications, that are gradually moving from traditional phone calls to be mostly through the Internet.
In May 2011, Google presented an open source project known as WebRTC [1], which pretended to introduce real-time communications through the browser and change our perception of the web. The fact that we could connect browsers and share audio and video streaming mean a huge innovation regarding the telephony and conference industry. But this is just the beginning, the ability for JavaScript developers to be able to work with the user local camera and microphone stream, or other user available user media (ie. Screen sharing or local files), opens a new range of possibilities and creates a more dynamic web that allow applications interact with the user in real time through voice, gestures, and a lot new options.

Until now we could get something like these functionalities in the browser with the help of plugins like flash, but this brought problems like crashes or constant updates, which resulted in a low quality perception in the eyes of the consumer.
On the other hand, it is hard to talk about video conferences without thinking about Skype, *What advantages can WebRTC present against Skype?*
In the first place, Skype is an application, it's a program that requires an installation in a machine and offers limited features. However, WebRTC is a set of tools available to everyone, that allows web developers to take advantage of their web programming knowledge (HTML, CSS and JavaScript) to add real time media communications to their creations.
This brings us two huge advantages:

- WebRTC based applications doesn't require of any plugin, just access the site though the browser.
- The fact that is an open source technology, unleashes the creativity of the developers to bring new ideas. Applications

based in this technology may mean a step forward in fields from gaming to e-learning or online streaming in the browser.

According to the arguments above, the motivation for this thesis has been to explore the potential of WebRTC technology to support the development of multimedia communication applications. The following section describes this objective in detail, as all sub-objectives that come from it.

C.2. OBJECTIVES

The main objective for this bachelor thesis is to analyze, from a practical perspective, the support brought by WebRTC technology to design, develop and deploy multimedia communication applications. To do this, the development of a multiparty conference application, that support exchange of audio, video, and text messages between the participants.

In line with this main objective, the following sub-objectives are defined:

- Analyze the state of art, reviewing its technical updated information about WebRTC operation, architecture, protocols, standards and tools, such as the most popular implementations.
- Assess the real possibilities to develop a WebRTC based product, considering difficulties, obstacles and facilities that we face in the current scenario.
- Explore the potential use cases of this technology.
- Reach a solid conclusion and make an assessment of the usefulness and maturity of the WebRTC technology.

C.3. DESIGN AND PLATFORM REQUIREMENTS

As mentioned in chapter 1, the principal objective of this thesis is to explore the WebRTC potential for developing and deploying multimedia communication applications. To achieve this goal, we will design develop and deploy a multiparty videoconference application with text chat.

The platform will follow a Star topology, similar to the triangle topology but with multiuser support. The general architecture is shown in the Figure 51 above:

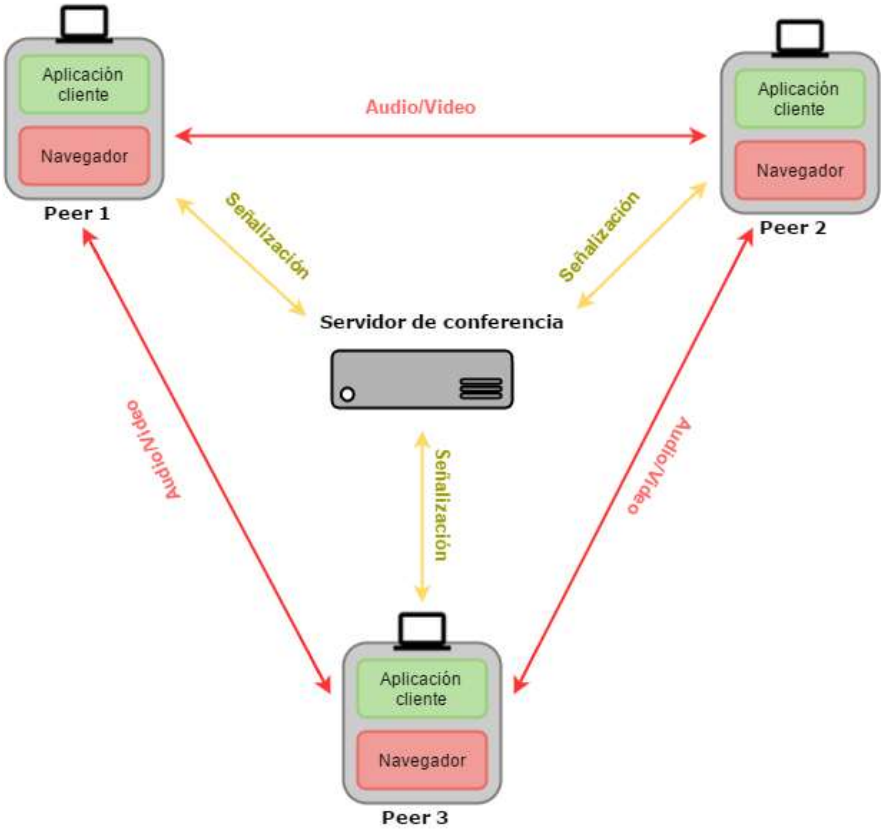

Figure 51:Application architecture for three participants

The service will have a client application that will run in the user browser with the support of WebRTC, that will be enabled in that browser. User will access the service through a conference server, introducing a URL in the browser. This server will exchange signaling information with the client allowing to setup a conference between all the users that introduce the same URL.
Functional requirements of the platform are described above:

- Star topology
- Support for multiuser connection. Each participant must send and receive information with the others and the conference server.
- Support for real-time Audio and Video Communications between connected participants.
- Text Chat support
- Signaling using WebSockets. Each participant will be able to setup a bidirectional connection with the conference server. This

connection will be used to exchange SDP packets and ICE candidates.

C.4. IMPLEMENTATION

This chapter will describe the operation and design of the actual platform. It will be separated in two blocks: Client application and conference server.
Figure 52 above shows the general architecture of the developed platform:

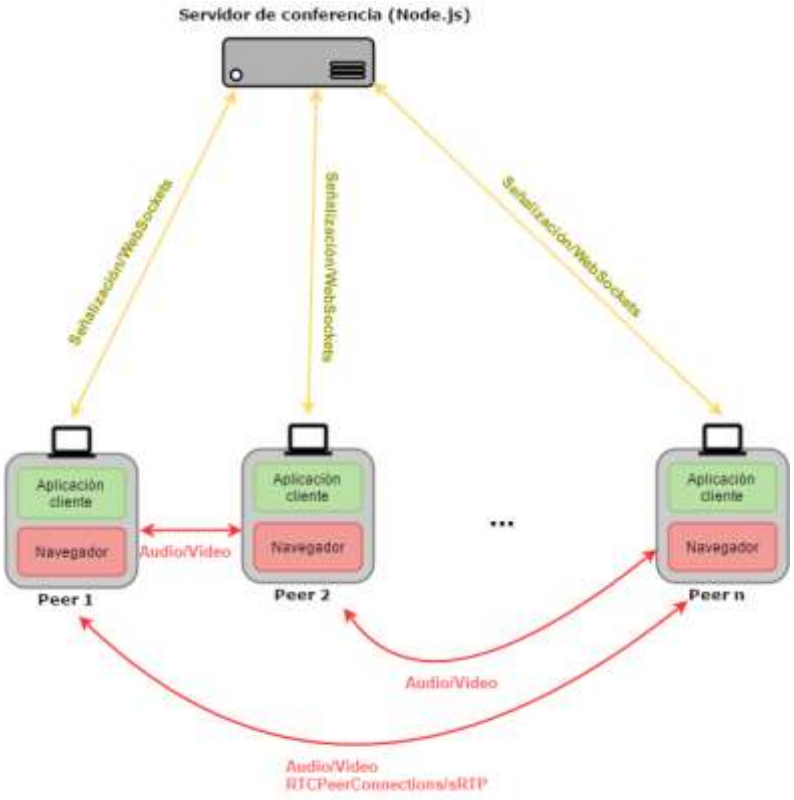

Figure 52: Platform general architecture

The application operation is simple, when a host access the web page a TCP socket opens between the browser and the server. This socket will work as a signaling channel transporting SDP messages and ICE candidates between the participants of the multimedia session.

Every time a new participant access the service, it will start a signaling dialog between the already connected participants, resulting in a RTC connection between all of them.

C.4.1. Client application

This section will contain the design and operation of the client application that will run in the user browser.

Graphical structure of the application is basically what the user sees when using the platform. Figure 53 shows the general graphic interface during a three user conference from the Client 1 perspective:

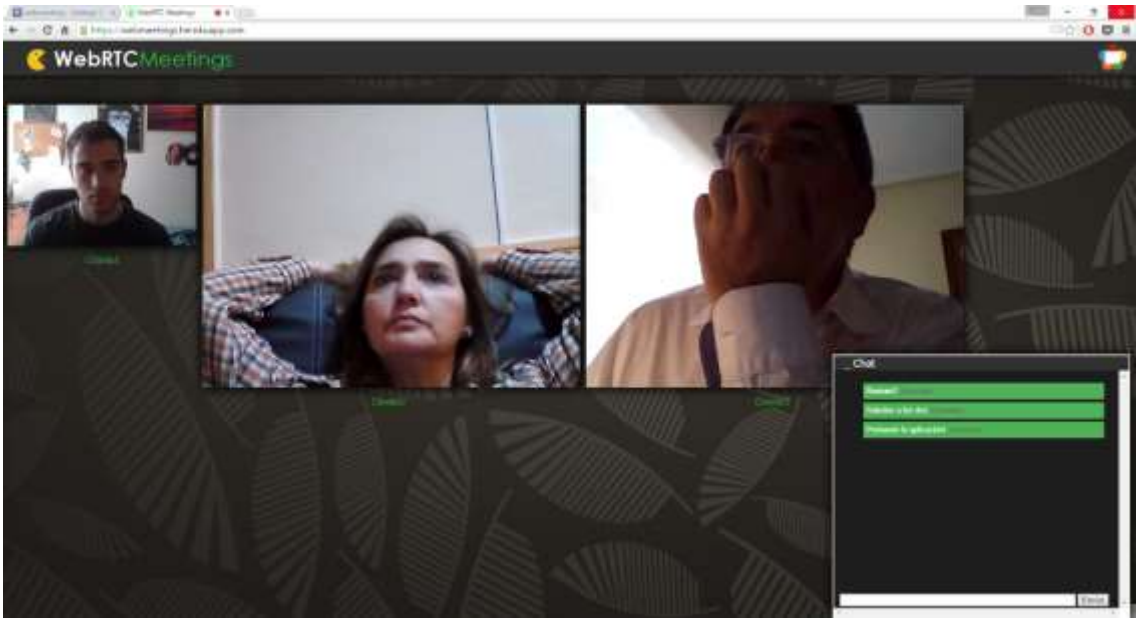

Figure 53:Graphical interface during a three user conference

Top left section, under the application logo, is the local video section. This contains the local multimedia stream obtained from the user's webcam and microphone. Under the video there's an id tag that works as a nickname.

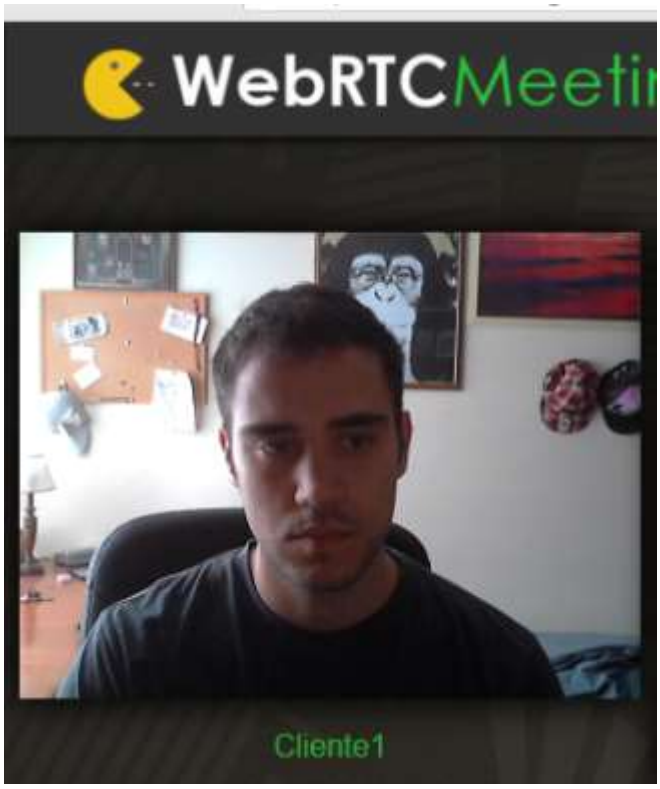

Figure 54: Local video

As other users access the URL, their video appear in the remote video section, at the right hand of the local video and with bigger dimensions.

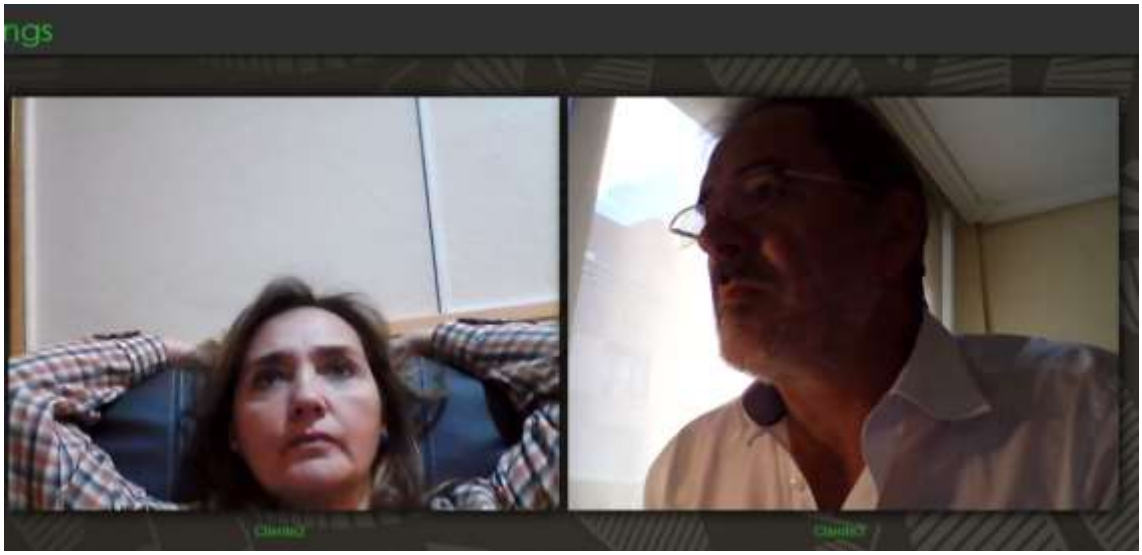

Figure 55: Remote Video

Bottom right corner shows the text chat section that shows and hides when the user clicks it. Chat messages are contained in green

bubbles and there is a violet tag with the nickname of the message sender near it.

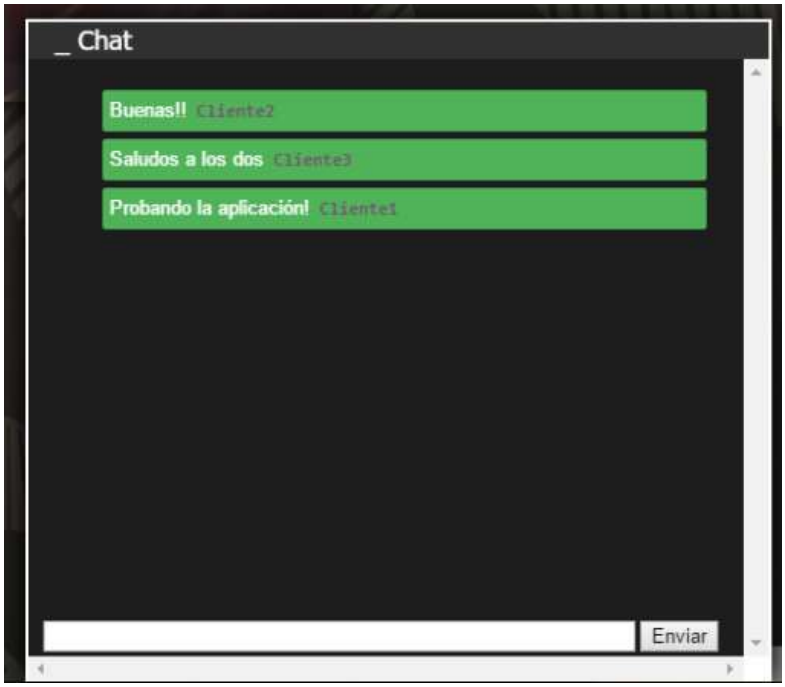

Figure 56: Text Chat

C.4.2 Conference server

Conference server's job is based on sending and receiving messages with the clients and activating certain events depending on the message type.
Messages can show the following types:

- Control messages: This messages are shown with certain user presence events, like when a new participant access the platform or when the participant abandons it.

- SDP Messages: In order to establish a multimedia session correctly, the SDP offer/answer dialog must be completed successfully. Users send the server these SDP messages as text strings with a specific destination and server works as a middle man.

- ICE Candidates: For a correct session establishment is also needed for the participants to exchange ICE candidates with IP addresses and ports.

- Text Messages: Chat messages are also sent to the server, who works as a middle man and redirects the received messages to all the connected users.

C.5. TESTING

This section will show the application network traffic using the WireShark tool. The testing topology is represented in the Figure X above, tests will be performed in a local network with private IP addresses and the conference server hosted in the heroku cloud application platform.

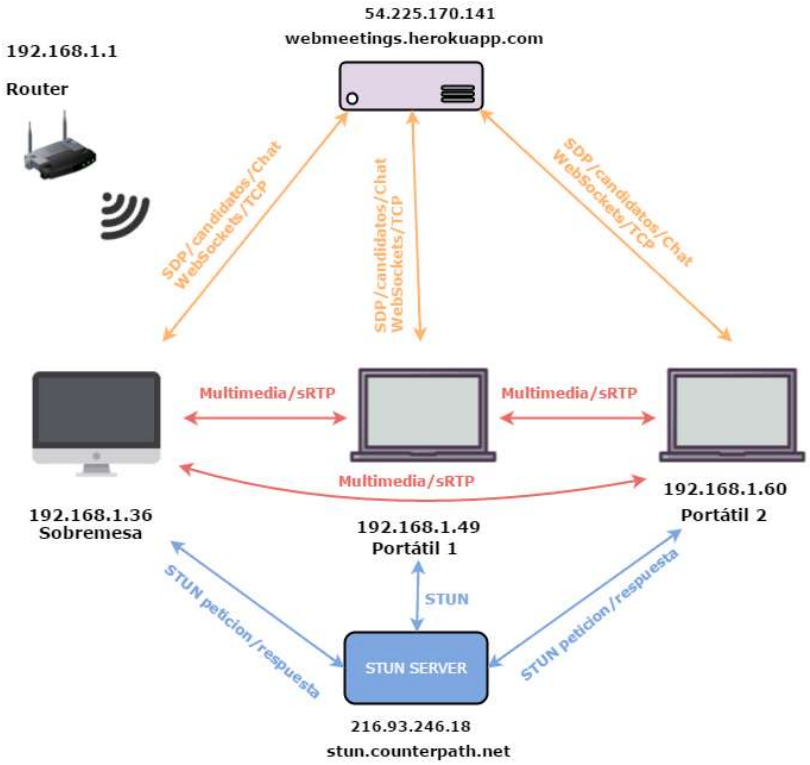

Figure 57: Testing Architecture

Figure 57 shows a Wireshark screenshot containing the bidirectional TCP connection between the host and the conference server.

54.225.170.141	192.168.1.36	TCP	60 443 → 52736 [ACK] Seq=1 Ack=453 Win=97 Len=0
54.225.170.141	192.168.1.36	TCP	60 443 → 52737 [ACK] Seq=1 Ack=505 Win=87 Len=0
54.225.170.141	192.168.1.36	TCP	60 443 → 52733 [ACK] Seq=1 Ack=485 Win=83 Len=0
192.168.1.36	54.225.170.141	TCP	54 52735 → 443 [ACK] Seq=472 Ack=324 Win=259 Len=0
192.168.1.36	54.225.170.141	TCP	54 52736 → 443 [ACK] Seq=453 Ack=171 Win=260 Len=0
192.168.1.36	54.225.170.141	TCP	54 52738 → 443 [ACK] Seq=491 Ack=325 Win=259 Len=0

Figure 58:Socket between the client and the server

STUN messages are between the host and the STUN server are shown in Figure 58. Host does a stun petition to know his public IP address and STUN server responds with the requested information:

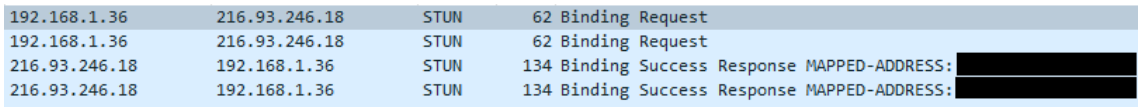

Figure 59: Stun petition

When the SDP dialog and ICE candidates exchange are completed, the multimedia session starts. Figure 59 show the RTP packet exchange between the three participants of the conference.

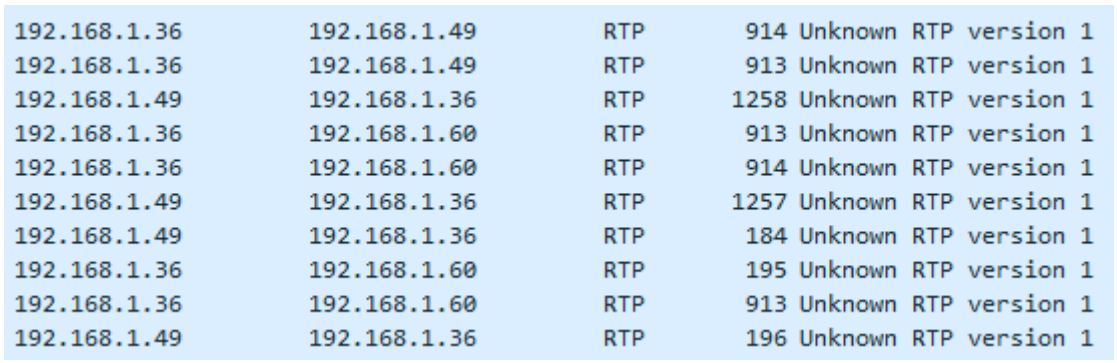

Figure 60: RTP Session

C.5. Conclusions and future lines

C.5.1 Conclusions

The development and deployment of a WebRTC based platform, as well as the research and knowledge of its capabilities, has demonstrated the great potential of this technology in the upcoming future.

The possibility for any developer to design and deploy a real-time communications service, coupled with the fact that network connected devices are growing exponentially, can result in a revolution in traditional communications and the way we perceive the web.

This new approach of the web could bring innovations in day to day services, such as real-time virtual medical consultations, or interactive online learning.

However, for this to happen, WebRTC still needs to improve certain aspects, including support for most popular platforms, a

significant increase of available codecs, and a clearer and handy documentation.

C.5.2. Future work

The following features and enhancements can be added to the application in the future:

- Using the RTCDataChannels API and a few graphical adjustments, we can build a File Transfer System so the conference participants can share files in real time.
- Screen sharing functionalities for the videoconference participants. This feature was intended to be included in the current platform, but the referred documentation on how to make it was not working without third party plugins.
- Security improvements, such as a login form that requires username and password. This information could be used as a user nickname in the corresponding video and chat messages.

These improvements will also help to explore the WebRTC potential to provide other multimedia services different as the ones exposed in this thesis, completing the study started in this project.

ANEXO D. CONCLUSIONS AND FUTURE LINES

This chapter will result in a final assessment of the project. The conclusions on the WebRTC technology will be presented, assessing its usefulness and maturity. Also obstacles and facilities during the development of the platform will be discussed. It will also contain the potential use cases of WebRTC in general and the future lines of work of the developed application.

D.1. Maturity of WebRTC

The development and deployment of a WebRTC based platform, as well as the research and knowledge of its capabilities, has demonstrated the great potential of this technology in the upcoming future.

The possibility for any developer to design and deploy a real-time communications service, coupled with the fact that network connected devices are growing exponentially, can result in a revolution in traditional communications and the way we perceive the web.

This new approach of the web could bring innovations in day to day services, such as real-time virtual medical consultations, or interactive online learning.

However, for this to happen, WebRTC still needs to improve certain aspects, including support for most popular platforms, a significant increase of available codecs, and a clearer and handy documentation.

D.2. Obstacles and facilities while developing the platform

As expected, main objectives exposed at the beginning of this document have been achieved. We have worked with Google open source tools developing and deploying a functional videoconference platform with support for multiple users.

Although WebRTC tools and APIs are accessible for anyone, documentation and tutorials on how to develop applications based on this technology are limited and they can sometimes be confusing. This

is also because the technology is not settled yet and capabilities are changing quite often.
Another important limitation when trying to become a WebRTC developer is that is complicated to create a product with the support of several browsers at once. At the moment Google Chrome is the most optimized browser for WebRTC development.

The short range of options for hosting real-time based applications without the requirement of external servers can also be considered a major obstacle, as it's required for the server to interact with the browser and make the signaling server functions. There are options like heroku that allows to perform this tasks, but its limitations cause stability problems and applications are difficult to become solid and reliable.

D.3. Future work

The following features and enhancements can be added to the application in the future:

- Using the RTCDataChannels API and a few graphical adjustments, we can build a File Transfer System so the conference participants can share files in real time.
- Screen sharing functionalities for the videoconference participants. This feature was intended to be included in the current platform, but the referred documentation on how to make it was not working without third party plugins.
- Security improvements, such as a login form that requires username and password. This information could be used as a user nickname in the corresponding video and chat messages.

These improvements will also help to explore the WebRTC potential to provide other multimedia services different as the ones exposed in this thesis, completing the study started in this project.